Review of: Simultaneously Structured Models With Application to Sparse and Low-Rank Matrices

Andersen Chang [1]

1. Introduction

One of the most common goals of data science and machine learning is to recover a true signal from noisy observations. In fact, one might say that this is what the whole foundation of statistics is built on. In order to do this, often times it is assumed that this true signal follows some set of structural constraints, such as sparsity, low-rankness, time consistency, or blockedness. This helps to simplify the model for ease of interpretation, prevent overfitting, and potentially reduce the number of samples required to find an estimate. While it is often more intuitively simple to pose this as a constrained optimization problem, in practice, the structural constraints are commonly imposed by using a penalized objective function, i.e.

$$\hat{\theta} = \underset{\theta \in \Theta}{\arg\min}\, \mathcal{L}(\theta|A) + \lambda p(\theta)$$

where $\mathcal{L}(\theta)$ is a loss function between the observed and predicted responses, $p(\theta)$ is a non-negative function that imposes adherence to the desired structure, and λ is a hyperparameter that controls the balance between regularization and prediction accuracy. In more complicated models and specific applications, multiple structures can be simultaneously imposed on a single estimated signal. For example, (Chandrasekaran et al., 2010) utilizes a sparse plus low rank structure in matrix decomposition in order to fit latent variable graphical models, which can be used in contexts such as fMRI and calcium imaging. (Grasedyck et al., 2013) utilizes multiple low rank structures in order to fit matrix tensor models with low Tucker rank, a model which is often used in physics and computational finance.

Of course, being the intellectual, brilliant, wizened, and principled data scientists that we all are, we are not satisfied with simply getting point estimates of model parameters from a black box model and algorithm and accepting them as correct. No, we also demand some characterization of the precision of the estimate as well. Typically, especially in the context of papers written for top tier statistics journals, this comes in the form of some kind of theoretical guarantee such as asymptotic consistency or finite sample error bounds. In the paper "*Simultaneously Structured Models With Application to Sparse and Low-Rank Matrices*," by Samet Oymak, Amin Jalali, Maryam Fazel, Yonina C. Eldar, and Babak Hassibi, the performance of recovering signals with multiple structural constraints is studied. Specifically, the authors derive theoretical probabilistic finite sample performance bounds in terms of sample complexity for both the general case of arbitrary simultaneously structured models as well as the specific case of a simultaneously sparse and low rank signal matrix.

One particular question the authors attempt to answer in the paper is whether using multiple simultaneous structures can reduce the sample complexity for obtaining a globally optimal estimate. It has been shown in many previous cases that adding one structural constraint to an unconstrained problem can reduce the sample complexity; one trivial example of this is adding the ℓ_1 Lasso penalty to a linear regression problem, which then allows one to get a consistent estimator even in the case where the number of observations is less than the number of model parameters. To this end, the authors show new theoretical results, as well as results from previous literature, for both convex and non-convex optimization problems corresponding to different simultaneously structured models.

2. Background

In this paper, the authors consider the situation in which we observe a signal $\mathbf{X} \in \mathbb{R}^{n \times d}$, possibly perturbed by a possibly random linear measurement matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$. (The exact perturbation can vary by context; for arbitrary cases, the operation is denoted as $\mathcal{A}(\mathbf{X})$. Specific definitions of $\mathcal{A}(\mathbf{X})$ are listed as necessary below.) The goal is to find the true signal $\mathbf{X}_0$, where it is assumed that $\mathbf{X}_0$ follows multiple simultaneous structural constraints. The authors note various specifically useful norms that can be added to loss functions in order to induce structure, including the ℓ_1 norm for sparsity, the $\ell_{1,2}$ norm for column sparsity, and the nuclear norm for low-rankness. Notably, these are considered in the literature to be convex relaxations of the ℓ_0, $\ell_{0,2}$, and $rank(\mathbf{X})$ penalties, respectively. The latter set of penalties allow for precise control of the desired spar-

[1]Department of Statistics, Rice University, Houston, Texas. Correspondence to: Andersen Chang <atc7@rice.edu>.

sity level or rank. However, they are nonconvex penalties, meaning that it is in general much more difficult to find optimal solutions algorithmically using these penalties. The former set of penalties, on the other hand, are convex and therefore simple to optimize via projected gradient descent methods such as proximal operators. The downside is that it can be difficult to control the exact desired structure of the model estimate due to the fact that these norms depend on the magnitude of the parameters.

To aid in the development and clarity of the theory for the paper, the authors also define some extra concepts and notations that are used in the rest of this paper (and literature review.) For a vector $\mathbf{x}$ or matrix $\mathbf{X}$, the normalized versions are denoted as $\overline{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|_2}$ and $\overline{\mathbf{X}} = \frac{\mathbf{X}}{\|\mathbf{X}\|_F}$, respectively. The set of all possible subgradients for an arbitrary norm $\|\|$ and vector $\mathbf{x}$ is denoted as $\partial\|\mathbf{x}\|$ and is called the subdifferential. The minimum absolute-valued correlation between the vector $\mathbf{x}$ and elements of a set S is defined as:

$$\rho(\mathbf{x}, S) = \frac{\|\overline{\mathbf{x}}\|}{\sup_{g \in S} \|g\|_2}.$$

For this context, the correlation quantity is particularly relevant for the set $S = \partial\|\mathbf{x}\|$, i.e. the subdifferential set mentioned above. The local Lipschitz constant for a given vector and norm is defined as $L = \sup_{g \in \partial\|\mathbf{x}\|} \|g\|_2$, i.e. the largest magnitude subgradient in the set of subdifferentials for a particular norm. Lastly, we define $\kappa = \frac{\|\overline{\mathbf{x}}\|}{L}$ (do not think of this as the typical condition number.)

2.1. Previous Work

The authors cite several recent previous works that have addressed the same issue of comparing the sample complexity for convex programs to their nonconvex counterparts. Specifically, these papers have looked at performance in different cases of structural constraints, including low-rankness, sparsity, and decomposition into low rank plus sparse matrices for square matrices (Candes & Plan, 2010; Candes et al., 2006; Wright et al., 2013). The results of these works are summarized in Table 1; in this particular setting, n is the number of rows and columns, k is the desired sparsity, and r is the desired rank, when applicable. This current paper looks to create a generalization of these previous results to arbitrary sets of structures and corresponding norms. As an example, it then provides results to a specific case that has not been analyzed before, namely the sparse and low-rank case.

The authors specifically note here that the convex recovery programs have already been shown to perform worse in these situations compared to the nonconvex versions. An important result of this paper will show that, from a probabilistic standpoint, for low rank and sparse matrices, the nonconvex recovery program will require a sample complexity of $O(r(k_1 + k_2)\log n)$ (where k_1 and k_2 are the row and column sparsity, respectively), while the convex recovery program has a sample complexity of $\Omega(rn)$, where $\Omega(.)$ denotes the best case complexity.

Table 1. Sample complexity results from previous lit for recovering k-sparse and/or r-rank signals from a signal matrix $X \in \mathbb{R}^{n \times n}$.

Structure	Nonconvex	Convex
Sparse	$O(k)$	$O(k \log \frac{n}{k})$
Low-rank	$O(rn)$	$O(rn)$
Low-rank + Sparse	N/A	$O((rn + k)\log^2(n))$

3. Main Results

In this section, I describe the important theorems that the authors present in the paper. Included here are results for the minimum number of samples required to ensure that there is some probability of correct recovery for the general case as well as for the sparse and low rank application. There are also more theorems, lemmas, and propositions that are presented in the paper, but these tend to either be special cases of the main theorems or statements used to prove the main theorems which are not formally proven themselves.

For the theories in the paper, the authors assume that the recovery a particular signal $\mathbf{x}_0$ with multiple simultaneous structures $S_1, S_2, \ldots, S_\tau$ can be thought of as the minimization of a function of a set of norms $h(\|\mathbf{x}\|_{(i)})$, $i \in 1, \ldots, \tau$ among all feasible points, i.e. where $\mathcal{A}(\mathbf{x}) = \mathcal{A}(\mathbf{x}_0)$. The authors define a vector-valued convex recovery program as the optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) = h(\|\mathbf{x}\|_{(1)}, \|\mathbf{x}\|_{(2)}, \ldots, \|\mathbf{x}\|_{(\tau)}) \\ \text{s.t.} \quad & \mathcal{A}(\mathbf{x}) = \mathcal{A}(\mathbf{x}_0) \end{aligned}$$

where h is convex and nondecreasing with respect to all $\|\mathbf{x}\|_{(1)}, \|\mathbf{x}\|_{(2)}, \ldots, \|\mathbf{x}\|_{(\tau)}$.

3.1. Deterministic Failure

Using the concept of Pareto optimality for convex functions, the authors explain that there exists a minimum number of measurements required to recover the true signal for the convex recovery programs because, without having a sufficient number of measurements, the true signal will not lie on the Pareto optimal curve. Armed with this concept, the author present Theorem 3.1, which quantifies the lower bound on the number of measurements required to prevent the deterministic failure of the vector-valued convex recovery program.

Theorem 3.1 *Suppose that we have*

$$\rho(\mathbf{x}_0, f(\mathbf{x}_0))\left(:= \frac{\|\overline{\mathbf{x}}_0\|}{\sup_{g \in \partial\|\mathbf{x}_0\|} \|g\|_2}\right) > \frac{\|\mathbf{A}\overline{\mathbf{x}}_0\|_2}{\sigma_{\min}(\mathbf{A}^T)}.$$

Then $\mathbf{x}_0$ is not a minimizer of the vector-valued convex recovery program.

Intuitively, what Theorem 3.1 tells us is that, for any particular problem, we need more samples if the subgradients of $f(\mathbf{x}) = h(\|\mathbf{x}\|_{(1)}, \|\mathbf{x}\|_{(2)}, \ldots, \|\mathbf{x}\|_{(\tau)})$ are closer to the direction that $\mathbf{x}_0$ is from the origin. This is in line with what we would expect geometrically: if the direction of the gradient is close to the direction that the signal is in, then a small step in the direction of the gradient will lead to a relatively larger change in the location of the estimate, meaning that it will be more numerically unstable. Thus, it makes sense that one would need more measurements for subgradients that are more aligned with the true signal.

The authors also emphasize that the inequality presented in Theorem 3.1 is deterministic, even when applied to random measurement ensembles. It is also mentioned here and later on in the paper that the bound from Theorem 3.1 may not be a tight bound or applicable in all cases. In particular, the bound is meaningless if $m > n$, since the matrix $\mathbf{A}^T \in \mathbb{R}^{m \times n}$ will have at least one linearly dependent column by definition, meaning that $\sigma_{\min}(A^T) = 0$ and that the right side of the inequality is infinite.

3.1.1. Upper and Lower Bounds

Using the general result of Theorem 3.1, one can derive specific bounds on the number of measurements required depending on the problem. Here, the authors bound the inequality of Theorem 3.1 in two separate ways. The left side of the inequality can be upper bounded in order to find the minimum value of $\frac{\|\mathbf{A}\bar{\mathbf{x}}_0\|_2}{\sigma_{\min}(A^T)}$ needed to guarantee no deterministic failure, or the right side of the inequality can be lower bounded to find the maximum value of $\rho(\mathbf{x}_0, \partial\|\mathbf{x}_0\|)$ needed to guarantee no deterministic failure. Propositions 3.1, 4.1, and 4.2, as well as Theorem 3.2, deal with the former of the two. Proposition 3.1 showing the bound for the general case and Theorem 3.2 showing the lower bound for (sub)Gaussian entries. Proposition 4.1 and 4.2 deal specifically with linear and quadratic Gaussian measurements, respectively.

Proposition 3.1 *Let $L_i = \sup_{g \in \partial\|\mathbf{x}_0\|_{(i)}} \|g\|_2$ be the local Lipschitz constant and $\kappa_i = \frac{\|\bar{\mathbf{x}}_0\|}{L_i}$ for the norms $i \in 1, 2, \ldots, \tau$ in a simultaneously structured model. Let $\kappa_{min} = \min\{\kappa_i\}$. We then have that:*

- $\rho(\mathbf{x}_0, \partial f(\mathbf{x}_0)) > \kappa_{min}$, $\forall f = h(\|\mathbf{x}\|_{(1)}, \|\mathbf{x}\|_{(2)}, \ldots, \|\mathbf{x}\|_{(\tau)})$.
- *Let $f = \sum \lambda_i \|\mathbf{x}\|_{(i)}$ for $\lambda_i > 0$. Let $\bar{\lambda}_i = \frac{\lambda_i L_i}{\sum \lambda_i L_i}$. Then $\rho(\mathbf{x}_0, \partial f(\mathbf{x}_0)) > \sum \bar{\lambda}_i \kappa_i$*

Theorem 3.2 *Let $\mathcal{M} \in \mathbb{R}^n$ be a closed convex set and $\boldsymbol{h} \in \mathbb{R}^n$ be an independent standard Gaussian vector. Define $\boldsymbol{D}(\mathcal{M})$ as the Gaussian distance of $\mathcal{M}$, i.e.$\boldsymbol{D}(\mathcal{M}) = E[\inf_{\boldsymbol{v} \in \mathcal{M}} \|h - v\|_2]$, and $\bar{\boldsymbol{D}}(\mathcal{M})$ is the standardized Gaussian distance Suppose $\mathbf{A}$ has independent standard Gaussian entries. The signal $\mathbf{x}_0$ will fail to be recovered with probability $1 - 10\exp(-\frac{1}{16}\min\{m_{low}, (1 - \bar{\boldsymbol{D}}(C))^2 n)\}$, where*

$$m_{low} = \frac{(1 - \bar{\boldsymbol{D}}(C))n\kappa_{min}}{100}.$$

Proposition 4.1 *Suppose the linear measurement matrix $\mathbf{A}$ has rows comprised of i.i.d sub-Gaussian vectors. Then, with probability $1 - 4\exp(-c_2 m)$ for $m \leq c_1 n$,*

$$\frac{\|\mathbf{A}\bar{\mathbf{x}}_0\|_2}{\sigma_{\min}(\mathbf{A}^T)} \leq \frac{2m}{n}$$

for $c_1, c_2 > 0, n = d^2$

Proposition 4.3 *Suppose we observe quadratic measurements $\mathcal{A}(\bar{\mathbf{x}}_0)$ of a signal $\mathbf{X}_0 = \mathbf{y}\mathbf{y}^T$, i.e. $\mathcal{A}(\bar{\mathbf{x}}_0)_{ij} = \mathbf{z}_i \mathbf{y}_i \mathbf{y}_j \mathbf{z}_j^T$ where $\mathbf{z}$ are independent standard Gaussian vectors or independent uniform vectors on a sphere of radius $\sqrt{d}$. Then, with probability $1 - 2ed^{-2}$ for $m \leq \frac{c_1 d}{\log d}$,*

$$\frac{\|\mathcal{A}(\bar{\mathbf{x}}_0)\|_2}{\sigma_{\min}(\mathbf{A}^T)} \leq \frac{c_2\sqrt{m}\log d}{d}$$

for $c_1, c_2 > 0, n = d^2$.

The important thing to note in both Theorems 3.1 and 3.2 is that the lower bounds on ρ are functions of the quantity κ_{min}. This means that, for any combination of norms in $f(\mathbf{x}_0) = h(\|\mathbf{x}\|_{(1)}, \|\mathbf{x}\|_{(2)}, \ldots, \|\mathbf{x}\|_{(\tau)})$, the lower bound of $\rho(\mathbf{x}_0, \partial f(\mathbf{x}_0))$ is at best the same as the lower bound of $\rho(\mathbf{x}_0, \partial h(\mathbf{x}_{\|}\|_{(i)}))$ for the single norm associated with κ_{min}. Thus, for the vector-valued convex recovery program, the number of measurements required to recover the true signal with greater than 0 probability for a simultaneously structured model is at best the same as the number of measurements required to recover the true signal with greater than 0 probability for a model with only one of the structures. From this, the authors conclude that adding multiple simultaneous structures to a convex recovery program does not in fact actually reduce the number of samples required to get a proper estimate of the true signal. This discovery motivates much of the work later on in the paper, in which the authors show that the performance of the corresponding nonconvex problems actually do improve on the performance.

Theorem 5.1 and Proposition 5.1 address the upper bounds of the right hand side of the inequality of Theorem 3.1; the former is for the general case, while the latter is specifically for sparse and low rank matrix recovery. Intuitively, the general upper bound from Theorem 5.1 implies that the correlation between the true signal and the subgradients for the

simultaneously structured models can not exceed the total distance covered by the weighted sum of the normalized subgradients themselves. Geometrically, this means that the worst case scenario is when the subgradient is aligned the signal itself, relative to the origin; thus, the correlation is bounded by the total distance of the weighted sum of normalized subgradients themselves.

Theorem 5.1 *Suppose the measurement matrix* $\mathbf{A}$ *has rows of independent standard Gaussians. Let* $f(\mathbf{x}) = \sum \lambda_i \|\mathbf{x}\|_{(i)}$. *Let* $\bar{\lambda}_i = \frac{\lambda_i \alpha_i^{-1}}{\sum \alpha_i^{-1}}$ *for* $\alpha_i > 0$. *Define*

$$m_{up} = \left(\sum \bar{\lambda}_i \boldsymbol{D}(\alpha_i) \partial \|\mathbf{x}_0\|_{(i)} \right)^2.$$

If $m \geq (\sqrt{m_{up}} + t)^2 + 1$, *then the convex recovery program will recover the true signal* $\mathbf{x}_0$ *with probability* $1 - 2\exp(-\frac{t^2}{2})$.

Proposition 5.1 *Suppose the linear measurement matrix* $\mathbf{A}$ *has rows of independent standard Gaussians, and the signal* $\mathbf{X}_0 \in \mathbb{R}^{d \times d}$ *is rank* $r < d$ *and has a* $k \times k$ *submatrix of nonzero entries. Let* $f(\mathbf{X}) = \lambda_1 \|\mathbf{X}\|_1 + \lambda_2 \|\mathbf{X}\|_*$, *where* $\lambda_1 = \beta\sqrt{\log(d/k)}, \lambda_2 = (1-\beta)\sqrt{d}$. *Then,* $\mathbf{X}_0$ *can be uniquely recovered with probability* $1 - 2\exp(\frac{t^2}{2})$ *whenever*

$$m \geq \left(2\beta k \sqrt{\log \frac{ed}{k}} + (1-\beta)\sqrt{6dr - +2d} + t \right)^2 + 1.$$

3.2. Application to Sparse and Low Rank Matrices

Here, the authors apply the general results from above to the case of recovering a true signal which with simultaneously sparse and low rank. To do this, they first define the general recovery program, which is similar to the convex version mentioned above:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) = h(\|\mathbf{x}\|_{(1)}, \|\mathbf{x}\|_{(2)}, \ldots, \|\mathbf{x}\|_{(\tau)}) \\ \text{s.t.} \quad & \mathcal{A}(\mathbf{x}) = \mathcal{A}(\mathbf{x}_0). \end{aligned}$$

For the general program, it is no longer assumed that h is convex with respect to all norms; this is done in order to use the concept for both a convex and nonconvex representation of the simultaneous sparse and low rank recovery problem. Theorem 3.3 below applies Theorem 3.1 for this problem. It presents results for three different scenarios: one for a general matrix structure, one for a positive semi-definite matrix where the signal is generally sparse, and one for a positive semi-definite matrix where the signal is specifically column sparse. For reference, the norms used and the measurement requirements for the different convex recovery programs to recover the individual structures mentioned in Theorem 3.3 are listed in Table 2. The full derivations for Theorem 3.3 can be found in the paper.

Theorem 3.3 *Say we want to recover a signal* $\mathbf{X}_0 \in \mathbb{R}^{d_1 \times d_2}$. *Let* k_1 *and* k_2 *be the desired row and column sparsity, respectively, and let* r *be the desired rank. Also, let* $\lambda_1, \lambda_2, c_1, c_2 > 0$. *Then we have:*

A. General model:

(a) Let $f(\mathbf{X}) = \|\mathbf{X}\|_{1,2} + \lambda_1 \|\mathbf{X}^T\|_{1,2} + \lambda_2 \|\mathbf{X}\|_*$ *and* $m_0 = \min\{d_1 k_2, d_2 k_1, (d_1 + d_2) r\}$. *Then the general recovery program will fail to recover* $\mathbf{X}_0$ *with probability* $1 - \exp(-c_1 m_0)$ *for* $m \leq c_2 m_0$.

(b) Assume that $f(\mathbf{X}) = \frac{1}{k_2}\|\mathbf{X}\|_{0,2} + \frac{1}{k_1}\|\mathbf{X}^T\|_{0,2} + \frac{1}{r}$*rank*$(\mathbf{X})$ *and let* $m_0 = \max\{(k_1 + k_2) r, k_1 \log(\frac{d_1}{k_1}), k_2 \log(\frac{d_2}{k_2})\}$. *Then the general recovery program will uniquely recover* $\mathbf{X}_0$ *with probability* $1 - \exp(-c_1 m)$ *for* $m \geq c_2 m_0$.

B. PSD, $\ell_{1,2}$ *norm:*

(a) Let $f(\mathbf{X}) = \|\mathbf{X}\|_{1,2} + \lambda_1 \|\mathbf{X}\|_*$. *Then the general recovery program will fail to recover* $\mathbf{X}_0$ *with probability* $1 - \exp(-c_1 r d)$ *for* $m \leq c_2 r d$.

(b) Let $f(\mathbf{X}) = \frac{2}{k_1}\|\mathbf{X}\|_{0,2} + \frac{1}{r}$*rank*$(\mathbf{X})$. *Then the general recovery program will uniquely recover* $\mathbf{X}_0$ *with probability* $1 - \exp(-c_1 m)$ *for* $m \geq \max\{rk, k \log dk\}$.

C. PSD, ℓ_1 *norm:*

(a) Let $f(\mathbf{X}) = \|\mathbf{X}\|_1 + \lambda_1 \|\mathbf{X}\|_*$ *Then the general recovery program will fail to recover* $\mathbf{X}_0$ *with probability* $1 - \exp(-c_1 m_0)$ *for* $m_0 \leq c_2 \min\{\|\bar{\mathbf{X}}_0\|_1^2, \|\bar{\mathbf{X}}_0\|_*^2 d\}$.

(b) Let $f(\mathbf{X}) = \frac{1}{k_1^2}\|\mathbf{X}\|_{0,2} +$ *rank*$(\mathbf{X})$ *and rank*$(\mathbf{X}_0) = 1$. *Then the general recovery program will uniquely recover* $\mathbf{X}_0$ *with probability* $1 - \exp(-c_1 m)$ *for* $m \geq c_2 k \log \frac{d}{k}$.

For all parts of the statements in Theorem 3.3, subpart (a) concerns the probability of failure for a convex characterization of the sparse and low rank problem if a minimum number of measurements is not met, while subpart (b) shows the probability of unique recovery under a nonconvex characterization of the same problem if a minimum number of measurements is met. To summarize what the point of Theorem 3.3 is, all of the minimum measurement requirements

Table 2. Summary of individual structures for recovering signal $\mathbf{X}_0 \in \mathbb{R}^{d \times d}$ where $n = d^2$.

Structure	Norm	L	$\|\bar{\mathbf{x}}_0\| \leq$	$n\kappa^2 <$
k-sparse	ℓ_1	$\sqrt{n}$	$\sqrt{k}$	k
k-column sparse	$\ell_{1,2}$	$\sqrt{d}$	$\sqrt{k}$	kd
Rank r	ℓ_*	$\sqrt{d}$	$\sqrt{r}$	rd

to uniquely recover the true signal with high probability in the nonconvex cases are less than the minimum measurement requirements to not fail to recover the true signal with high probability in the convex cases. Specifically, in all three situations listed here, once the dimensionality of observations and/or signal is sufficiently large, the nonconvex approaches have a general sample complexity that scales logarithmically with the number of dimensions, while convex approaches have a best case sample complexity that scales linearly with the number of dimensions. Thus, it is clear for this particular problem that the nonconvex regime with perform much better than its convex counterparts, all other things being equal. (Whether or not all other things are equal is discussed in section 5 of the literature review.)

4. Simulation Results

A wise man once said, "Theory and practice are the same in theory, but not in practice." Thus, it is probably a good idea to confirm that what is seen in empirical results matches what the theory says should happen. In this paper, in order to confirm some of their theoretical results, the authors perform a simulation study on the convex program for recovering a (normalized) sparse and low rank signal matrix. In all of these simulations, the authors attempt to find a true signal for $d \times d$ matrices with $k \times k$-sparse value generated from $r < d$ i.i.d Gaussian distributions; they choose $k = 8$ and $r = 1$ for the setting.

From Theorem 3.3 part (B), the best case required sample complexity required to recover a sparse and low rank positive semi-definite matrix, using an $\ell_{1,2}$ norm for sparsity, is on the order of $\Omega(rd)$ (where $n = rd = d$). The simulation results are shown in Figure 1. In the figures, the dark areas show the empirical region of high failure of recovery. From the simulation study, it appears that the number of samples required to get recovery with high probability increases linearly with $n = rd$, which is in line with what the theory from the paper says.

Similarly, Figure 2 shows the empirical results for recovering a a sparse and low rank positive semi-definite matrix, this time using an ℓ_1 norm for sparsity. The additional green line on the plot shows the border for the empirical 95% failure boundary. From Theorem 3.3 part (C), the sample complexity should be on the order of $\Omega(\min(k^2, d))$. Again, the dark region shows the area of high probability of failure. Here, the required sample complexity appears to grow approximately linearly with $n = rd$ until it reaches a certain point, after which the sample complexity appears to grow on the order of $O(\log d)$. The first part of these results matches what the theory says. However, the second part does not, as the theory says this should be constant given a predetermined value of k, while the simulation says that it appears to grow logarithmically with d. The authors posit that this is due to the sample complexity required to minimize and ℓ_1 norm, which eventually dominates the constant term.

Figure 1. Simulation results for Theorem 3.3 part (b).

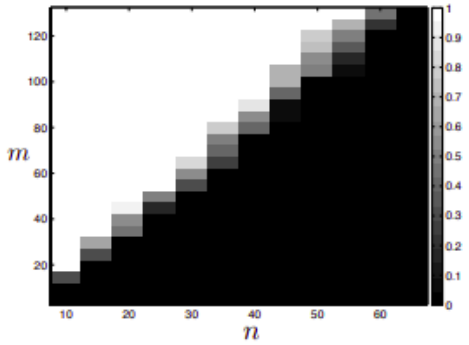
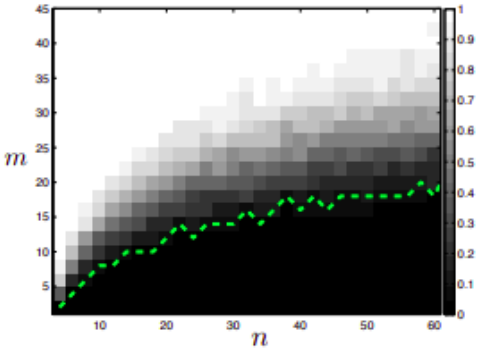


Figure 2. Simulation results for Theorem 3.3 part (c).



5. Discussion

Overall, I thought that the paper had some compelling theoretical results. In particular, it showed that, from the perspective of sample complexity, there may be more merit in using nonconvex formulations to induce multiple simultaneous structural constraints on an estimated signal as opposed to a convex one. A lot of the work in this paper is related to the research work that I currently do. In the past, I have used sparse plus low-rank decomposition methods to fit latent variable graphical models, which used the convex formulations of the sum of an $\ell 1$ norm and a Frobenius norm as the regularization penalties. Knowing that, in theory, using the nonconvex formulation may help reduce sample convexity may be a good motivation to look at different potential formulations. Also, as someone who has been brought up to predominantly use ℓ_1 norms to induce sparsity and nuclear norms to induce low-rankness, it has been interesting to me to see people argue for using a nonconvex ℓ_0 norm (or even the ℓ_2 ridge norm) for inducing sparsity.

From a technical writing standpoint, I thought that the authors could have done a better job in terms of organizing the structure of the paper to be more cogent to the reader. The theorems and propositions could have been put in a more logical order so that related ideas could be presented in a more continuous fashion. For example, the upper bounds of the inequality for Theorem 3.1 is for some reason shown in sections 4 and 5. Also, the application of the theorems to the specific sparse and low rank application are interspersed throughout the paper along with the corresponding theorems; these may be better served being put in a single standalone section. Additionally, it seems odd to me that the proofs for theorems in the 3rd section were put in sections 6 and 7, which comes after all the sections for the theory but before the numerical experiment and discussion section. I feel like these either belong next to the corresponding theorems or in an appendix after the main body of the paper, as it currently makes the paper difficult to browse through to find certain proofs or results. Unless, of course, they are trying to hide the simulation and discussion sections of the paper so that people will not read them for fear of the great shame it will bring to the authors. In which case, I completely understand, though I still do not condone the action.

One potentially interesting alternative to nonconvex penalties brought up by the authors in the discussion section of the paper is the creation of single atomic norms for inducing multiple simultaneous structural constraints, as first explored and characterized in (Chandrasekaran et al., 2012). That paper provides a framework for constructing new, single norm constraints for multiple simultaneous convex structures by utilizing the geometry of the convex hulls created by each of the individual structural constraints. The authors of this current paper mention that, while they have shown that the sample complexity for the sum multiple convex norms is at best on the same order as the smallest sample complexity of the individual norms, there is a possibility that using a single convex norm as a substitute could actually prove to break the theoretical lower bound for simultaneously structured models with sums of convex norms that they have presented here. This could be an interesting alternative to pursue rather than having to deal with nonconvex optimization algorithms, as the latter presents a computational problem as we will discussed below.

The authors also mention a couple of other interesting directions of research that can follow up from this paper. Firstly, this paper takes an approach of quantifying the probability of unique recovery (or failure to do so) given the number of measurements made. However, often times we already have a set of data and can not change the number of measurements we have. Thus, instead of calculating a probability of failure to determine how many measurements we want to take, it might be of more use to derive error or confidence bounds for the estimate given by a simultaneous structured model given the number of measurements observed. Another topic that could be of interest would be to apply the theories in the paper to other problems where the concept of degrees of freedom or number of measurements is not as clear cut. For example, the authors posit the idea of using their theories on the sparse PCA problem. Unlike the straightforward signal recovery problem studied in this paper, this problem does not correspond to having one measurement per observation in the data. Thus, a potentially new framework would need to be created in order to find the corresponding minimum necessary observations. Other more difficult problems that this could be applied to include latent variable models or non-independent observations.

As mentioned previously in this literature review, a wise man once said, "Theory and practice are the same in theory, but not in practice." In this particular paper, the authors do not spend much time addressing the issue of how to find solutions to the nonconvex programs algorithmically. While there may be solutions for specific cases of certain combinations of nonconvex norms, such as interior point solves for the sparse and low rank case, it may be difficult to find an efficient and generally applicable algorithm that can work on any arbitrary set of simultaneous structural constraints. Thus, if there is no reliable and efficient way to estimate the solution to a particular nonconvex program for a certain set of simultaneous structures, the reduced sample complexity theoretically required to get a correct estimate of the true signal for the nonconvex regime compared to the convex one may not outweigh the time complexity required algorithmically.

One possible algorithm which the authors mention is to use an Alternating Directions Methods of Multipliers (ADMM) algorithm, a method that utilizes Lagrangian dual spaces in order to do blockwise iterative updates for optimization. To the authors' credit, the ADMM algorithm has been used in a wide variety of other settings to perform optimization for otherwise difficult problems. However, this by itself may not be a good enough solution. While this has been shown to have good convergence properties for nonconvex and nonsmooth optimization problems in the case of a two-block problem (Wang et al., 2019), which in this case would correspond to a single norm structural constraint, other papers have shown that the ADMM algorithm may diverge in the case of multi-block problems (Chen et al., 2016), which can arise in this context by having multiple norms for simultaneous structures. Thus, unless one can create a single norm that encapsulates all of the desired structural constraints (something akin to a nonconvex version of what would be created under the framework of (Chandrasekaran et al., 2012)), the theory in the paper may not actually be useful in practice.

References

Candes, E. and Plan, Y. "tight oracle bounds for low-rank matrix recovery from a minimal number of random measurements. *IEEE Transactions on Information Theory*, 57(4):2342–2359, 2010.

Candes, E., Romberg, J., and Tao, T. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inform. Theory*, 52:489–509, 2006.

Chandrasekaran, V., Parrilo, P. A., and Willsky, A. S. Latent variable graphical model selection via convex optimization. *IEEE 2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1610–1613, 2010.

Chandrasekaran, V., Recht, B., Parrilo, P. A., and Willsky, A. S. The convex geometry of linear inverse problems. *Foundations of Computational mathematics*, 12(6):805–849, 2012.

Chen, C., He, B., Ye, Y., and Yuan, X. The direct extension of admm for multi-block convex minimization problems is not necessarily convergent. *Mathematical Programming*, 155(1-2):57–79, 2016.

Grasedyck, L., Kressner, D., and Tobler, C. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.

Wang, Y., Yin, W., and Zeng, J. Global convergence of admm in nonconvex nonsmooth optimization. *Journal of Scientific Computing*, 78(1):29–63, 2019.

Wright, J., Ganesh, A., Min, K., and Ma, Y. Compressive principal component pursuit. *Information and Inference: A Journal of the IMA*, 2(1):32–68, 2013.

Applied Machine Learning for Materials Science

Andrew Hitt [*1]

Abstract

In recent years, the use of machine learning within materials science has been on the rise. While some of these applications of machine learning are simply post-experimental analysis, other implementations seek to offer unique solutions to problems within the field of materials science. Some of these uses of machine learning include improving both the discovery and synthesis of novel compounds with desirable properties, expediting the application of conventional analytical techniques, and providing alternative methodologies for use in certain domains. Overall, the interfacing of machine learning into materials science has great promise for advancing all stages of the materials design process, from discovery to synthesis to evaluation to analysis.

1. Introduction

Materials science is, in the most general sense, meta-engineering. Although it relies heavily on mechanical engineering, with which it is often associated, in recent years, materials science has become a full-fledged interdisciplinary field, sitting at the intersection of mechanical, chemical, and electrical engineering. On a more technical level, materials science is the investigation of the three-pronged relationship between structure, processing, and behavior in materials. Due to its interdisciplinary nature, materials science often requires the exploration of alternative solutions to existing problems; one recently popular method for approaching problems in both materials science and other fields of physical science is the application of machine learning. In this review, the use of machine learning within materials science is discussed, with specific emphasis on the application of machine learning to address larger problems within the field.

Although both materials science and machine learning are well-established fields, the use of machine learning methodologies within materials science remains a relatively recent development. Despite this, machine learning has been rapidly adopted within the field as a tool to aid in the analysis of experimentally generated data for many different domains of research, including microstructural characterization (de Albuquerque, 2008), phase segmentation (Carrasquilla & Melko, 2017), and numerous other applications.

While it is unsurprising that the use of machine learning in materials is on the rise as scientific computing and data science become more popular, the use of more advanced machine learning techniques within the field are especially interesting. Linear regression, which has been used across virtually all quantitative domains for centuries since its development by Legendre and Gauss in the early 1800s, may technically be a supervised machine learning algorithm, but its use in materials science is not representative of the extent of more powerful algorithms, such as neural networks or support vector machines within the field. Furthermore, it is important to make a distinction between how the machine learning models are being used by researchers; some are used to supplement more conventional analysis methods, while others may be deployed more independently as an alternative to current techniques.

The use of machine learning within materials varies greatly depending upon the domain of research and the goals of the application of machine learning methodologies (Butler, 2018). While a machine learning model and a dataset of adequate size are all that is required to perform machine learning, doing so without an explicit goal in mind can lead to ultimately irrelevant results. This, in turn, means that uses of machine learning that are deployed to solve very specific problems are especially significant within the field.

This review covers four different novel uses of applying machine learning techniques to solve more general problems within the field, typically by demonstrating a proof-of-concept on a relevant experiment. The first example is the use of an unsupervised clustering algorithm to reduce noise in a volumetric dataset, substantially improving the subsequent analysis. The second example is the use of support vector regression to predict how materials behave while under circumstances outside of those that current materials science methodologies are capable of handling. The third example is the use of generative adverserial networks to improve efficiency in otherwise time-consuming microstructural simulations. The fourth example is the use of meta-analysis via natural language processing of thousands of published articles to generate an artificial intelligence to aid in the development of materials within certain constraints.

Collectively, these examples demonstrate that machine learn-

ing can be more than just a simple analytical tool; with proper application, machine learning can provide solutions to some of the largest problems within the field of materials science.

2. Discussion

Machine learning can be applied in a variety of different ways to address problems in materials science. Some of these uses might be smaller in scale, such as performing least-squares linear regression on data from a single set of experiments, while others may be rather significant, such as performing meta-analysis on all articles in a given domain to counteract the otherwise limited data resources. While machine learning models can be applied to almost any dataset of a suitable size, some of the particular interesting applications of machine learning are those that provide solutions to problems that cannot be addressed through conventional means.

2.1. Improving Analysis of Volumetric Data

The most obvious way that machine learning is used within mateirals science is as a tool to aid in post-experimental analysis. Like many other fields of science, research within materials science generates a sizable quantity of data and transforming that data into meaningful results can often be an immensely difficult and time-consuming process. As with simpler models, it is important to intelligently apply machine learning models to available data; any machine learning model and dataset of sufficient size can be combined, but that by itself does not mean that the trained model is useful beyond its novelty.

One problem of great importance within materials science is the analysis of volumetric (three-dimensional) data. Outside of a small amount of materials (graphene, thinfilms, carbon nanotubes, borophene, etc.), most materials have three-dimensional structures on both the micro- and the macroscale. Analyzing samples of these three-dimensional materials thus often requires the use of three-dimensional imaging techniques such as neutron scanning tomography or iterated scanning electron microscopy. Such imaging processes result in three-dimensional datasets, upon which more substantive analysis can then begin.

Analyzing volumetric data can be immensely time-consuming. Most of the three-dimensional imaging techniques produce datasets that require extensive amounts of post-processing before the actual analysis can begin. Tomography data, for example, can have significant noise due to the way that it is reconstructed (this is especially important for biomedical fields, where tomography is more prevalent), and iterated scanning electron microscopy can require image realignment the images to ensure that adjacent slices of a sample have not skewed one way or another. Beyond this, analyzing volumetric data is itself rather difficult, as many image processing techniques are not designed for three-dimensional applications in mind, and thus require custom implementations for use on volumetric data. Furthermore, image processing algorithms are often of moderate to high computational complexity; a kernel filter of size k deployed on an image of dimensions $x \cdot y$, for example, is $O(xyk^2)$, which in turn scales into $O(xyzk^2)$ for three-dimensional applications.

To address some of these problems with their own tomography data, one group of researchers applied machine learning techniques. While transfer functions methods are usable for smaller datasets, they do not scale efficiently for use on larger datasets. As noise reduction of some form is necessary to use for the data to be usable in subsequent analysis, they focused their efforts on utilizing the fact that considered spatial characteristics of each voxel in their tomography data. To do this, the authors employed an unsupervised learning technique called density-based clustering for applications with noise (DBSCAN) and use it to aid in the reconstruction of their neutron scattering tomography data (Hui & Liu, 2018).

DBSCAN is an algorithm that groups a set of points into clusters based on regional density (Ester, 1996). Under DBSCAN, all points in a dataset are assigned a label: core (if the point is within an arbitrary distance ε of many other points), reachable (if the point is not a core point but is within an arbitrary distance ε of at least one core point), or noise (if it is neither core nor reachable). The paper's innovation on this technique, beyond simply integrating DBSCAN with the tomography reconstruction, was to vary how many nearby points were needed for a point to be labeled "core" rather than "reachable" based on the intensity at that point.

This application of DBSCAN to reduce noise in the tomography data was shown to significantly reduce the noise present in the data and was able to generally improve the confidence in the results of subsequent analysis. The researchers also noticed that in addition to generally improving the reconstruction, the noise reduction also permitted for the identification of finer features in some of the large internal structures of their sample.

2.2. Predicting Material Behavior

Predicting material behavior is one of the most important problems in the field of materials science. Much of materials science is focused on the development and subsequent implementation of materials with very specific sets of properties for a given application, and thus being able to assess whether a given material fulfills all of the design criteria is paramount. The current standard for prediction of most ma-

terial properties is density functional theory (DFT), which simulates material behavior through applications of functional quantum chemistry. Density functional theory has been used to predict many complex material properties, including dipole moments, magnetic behavior, and molecular geometry (Langenaeker, 2003).

While density functional theory is a powerful tool that has greatly improved the field of materials science, it has several notable limitations. One of the most significant issues is that density functional theory is not optimized to predict non-equilibrium behavior, which renders it effectively unusable for situations where the material's behavior under plastic deformation is important. Furthermore, it has been demonstrated that the prediction of plastic deformation is further complicated by the multiple length scales involved (Zhao, 2004).

Collectively, these problems have forced researchers to investigate alternative methods for the prediction and development of materials that perform well under plastic deformation. Within the field of materials science, the term "hardness" is used to quantify a material's ability to resist localized plastic deformation (Zhang, 2001). Hardness can be tested experimentally through either scratch testing (using the Mohs scale) or indentation testing (typically using the Vicker's scale, although other testing methods exist). Generally, the value of a material's Vicker's hardness is a more intuitive descriptor than the value of the material's Mohs hardness, as Vicker's hardness uses a linear scale whereas Mohs uses an arbitrary scale, and is thus less quantitatively consistent.

Of particular interest are the "superhard" materials, which are defined as materials with a Vicker's hardness of greater than 40 gigapascals. Superhard materials have a wide variety of industrial applications, including as scratch-proof coatings (Phaal, 1991) and powerful abrasive agents (Lux & Haubner, 1994). Despite this variety of uses, there are substantial issues with the two predominant superhard materials (diamond and cubic boron nitride) that necessitate the development of new superhard materials (Kaner, 2005).

As density functional theory is ill-suited to the tasks of predicting hardness, researchers often are forced to resort to trial-and-error methods, which is incredibly costly in both time and financial resources; as a result of this significant cost, materials scientists often restrict their research into superhard materials to a limited class of materials known as intrinsic compounds (Tehrani, 2017), which include diamond and boron nitride along with other, typically non-metallic, compounds. In more recent years, researchers have also begun investigating the addition of high-valence transition metals to expand the region of exploration to include intermetallic compounds as well, but the development, synthesis and testing of these compounds remains expensive.

Thus, the discovery of new superhard compounds is plagued by issues that limit its progress; existing computational techniques are almost unusable for the purpose, and doing so experimentally is expensive with no guarantee of positive results despite the enormous cost. One solution to these problems was ultimately found in machine learning, through the application of support vector machines (SVMs) using data from two of the largest materials science databases: the Materials Project and Pearson's Crystal Database (Brgoch, 2018).

The support vector machines were trained on a test set from Materials Project that contained hundreds of descriptors of material composition, processing, and properties for over 2,500 different materials. The trained model was then deployed on more than 100,000 compounds listed on Pearson's Crystal Database, generating predictions for the bulk and shear modulus of each compound. Bulk and shear modulus are often used as proxies for Vicker's hardness due to their strong correlation across many different classes of materials (Liu & Cohen, 1989).

Taking the results from these predictions, the researchers then selected the ternary (3-element) and quarternary (4-element) compounds with the highest predicted elastic moduli and synthesized them. With these samples, they then tested the bulk modulus (via X-ray powder diffraction) to provide a direct comparison with predicted values, as well as the Vicker's hardness to quantify whether the materials selected demonstrated superhard behavior.

Ultimately, these researchers found that the machine learning model performed quite well across their data set, with agreement between the predictions of the support vector machine and recorded experimental data within 20%. Although the model struggled with predictions for certain classes of materials (such as compounds containing rare-earth metals), both of the materials predicted to have superhard behavior possessed Vicker's hardness values in excess of 40 gigapascals under some conditions, confirming superhardness.

2.3. Simulating Kinetic Processes

A third way that machine learning methods can be used is as a full replacement of existing techniques that prove to be inefficient or otherwise ill-suited to applications within certain domains.

In addition to interest in predicting the bulk properties of materials (such as hardness), there is also substantial research into the ways that matter moves throughout a material. While diffusion itself is important for many processes, such as semi-conductor processing, one of the most significant of these effects is grain evolution. Grain evolution is the process through which grains (small regions of continuous crystallinity within a larger sample) change over time in

an effort to minimize the energy of the system; during this evolution, grains can change their shape and even absorb smaller grains if the kinetics are favorable.

The reason why grain evolution is so interesting to materials scientists is that the size and shape distributions of the grains within a material (often referred to as the microstructure) can substantially impact a material's properties. Grain size can be shown to affect mechanical properties such as elastic modulus (Kim, 1999), electrical properties such as conductivity (Badwal & Drennan, 1987), and chemical properties such as surface reactivity (Santos, 2010). As the microstructure is capable of changing so many material properties, being able to predict the ways that microstructure may evolve is obviously incredibly important.

Generally, grain evolution refers to grain growth, as in most materials, a reduction in surface energy permits kinetically favorable changes in the microstructure such that small grains get slowly absorbed into bigger grains, when provided sufficient activation energy. The increase of average grain size can be favorable or not depending on the material and application. Notably, the Hall-Petch effect says that yield stress (a measure of the highest stress a material can handle before plastically deforming) decreases with increasing grain size, and thus for most mechanical applications, a smaller grain size is often preferable; this is not true at higher temperatures, where a small grain size increases the rate of Coble creep, a phenomenon by which vacancies diffuse through a material causing plastic deformation (Mohamed & Langdon, 1974).

Within materials science, and most fields of physical science, the industry standard for grain growth simulation is COMSOL Multiphysics (often abbreviated to just COMSOL). COMSOL is a combination of finite element analyzer and multiphysics simulation solver. COMSOL is also optimized to solve systems of coupled partial differential equations, which is the capability that makes it appealing for grain growth simulation.

While COMSOL does offer an efficient method for predict grain growth and evolution in materials, its use has a few substantial limitations that significantly reduce its efficacy. Running simulations in COMSOL can take incredibly long amounts of time (up to hours of computational time for seconds of simulation time), despite being quite optimized for this application. Beyond this, COMSOL's predictions for grain evolution can be heavily influenced by the choice of surface energy model, which can negatively impact confidence in the simulation results (Zaeem, 2011).

To address these problems, researchers have recently begun investigating the use of machine learning models to provide an alternative to the computationally expensive simulations (Tang, unpublished). To generate a dataset for training this machine learning model, thousands of Voronoi diagrams (graphs constructed by defining regions based on the closest point from a set of points) were generated. Short-term grain growth simulations were then run on these Voronoi diagrams in COMSOL to create pairs of images (initial and after ten seconds of simulation time).

Using this training dataset, a generative adverserial network (GAN) model was developed. Generative adverserial network models are comprised of two separate neural networks: a generator (which creates fake samples from random noise) and a discriminator (which tries to determine whether a given sample is real or generated). For the application of grain growth simulation, the both networks in the GAN were convolutional neural networks designed to upscale or downscale images to or from a single numeric value. Here, the generator is tasked specifically with simulating grain evolution from an initially provided image. Thus, the discriminator is similarly assessing whether the provided post-simulation image is from COMSOL or from the generator.

Preliminary results of this research are rather promising. After about 50 epochs of training on subsets of the training data, the generator was able to produce grain evolution patterns that were nearly identical to those produced in COMSOL, including on novel Voronoi diagrams not used at all in the training data. Admittedly, the generator struggled to produce accurate simulations for certain Voronoi diagrams, particularly ones where it still struggled with certain grain distributions with well separated regions of either small or large regions, but this could hopefully be improved upon with further development of the algorithms and training of the adverserial network.

2.4. Leveraging Domain Knowledge

One of the most notable problems in terms of the application of machine learning models or other data science techniques within materials science is that there is often limited data available. Materials science, more so than other fields like biosciences, is notable in that the datasets for materials are often small in comparison to other fields. The Open Quantum Materials Database (QGMD), which was created by the Wolverton group at Northwestern University and is often lauded as the largest materials science database currently in existence, contains data of approximately 637,000 materials; for comparison, a single sequenced human genome contains data for approximately 3 billion base pairs. Due to this, there is great interest in and extensive discussion within materials science on how to make the most of the relatively limited data available.

Currently, much of the effort into maximizing the value of experimental data is invested in the development of materials databases, such as the previously referenced Open Quan-

tum Materials Database, Pearson's Crystal Database, and the Materials Project. The core principles underpinning all of these efforts is that materials science, as a field of science based heavily on stochastic processes and well-understood physical models, is perfectly suited for the deployment of data science techniques, and these projects seek to provide datasets of sufficient size for significant results to appear when used for analysis.

One option for extending the use of research beyond its original article is meta-analysis, a technique that has been gaining popularity in recent years. While meta-analysis can be handled manually, as it is many literature reviews, its results significantly improve as the amount of articles analyzed increases. This has caused a push for automated meta-analysis techniques in recent years across all fields of science; one example of an automated meta-analysis platform is MetaCyto (Hu, 2018), which is optimized to perform meta-analysis on cytometry studies.

Within materials science specifically, the use of automated meta-analysis remains rather limited. The Materials Genome Initiative (MGI) has begun investigating meta-analysis as an option for high-throughput automated design of novel materials, and has used this framework for numerous different applications including batteries and thermoelectric materials. However, most of the results produced by the Materials Genome Initiative are more theoretical predictions of materials with desirable behaviors, rather than suggestions for how to synthesize a material with specific properties.

To address the lack of computationally generated synthesis methods, one group of researchers developed a natural language processing framework that was capable of analyzing thousands of articles about metal oxide synthesis (Olivetti, 2017). To do this, the researchers first compiled a database of over 12,000 articles about the synthesis of various metal oxide compounds, which were downloaded in PDF format before automated conversion to plain text. A classifier to distinguish which paragraphs were specifically about synthesis was constructed from a training set of several hundred manually labeled paragraphs and then deployed on the remainder of the articles to reduce the amount of text being analyzed. Relevant paragraphs were then parsed using natural language processing, specifically dependency parse trees, word tokenization, and part-of-speech tagging. Error correction (such as fixing the misrepresentation of the degree symbol ^{o} as a numerical digit) was performed.

After the dataset was generated and properly refined, it was then used to train several different machine learning models. The authors trained a support vector machine, a linear classifier, and a random guessing strategy on the data and then compared the performance of all three models. Unsurprisingly, the support vector machine performed muich better than the other two models (the linear classifier barely performed better than random guessing) on the set of arbitrarily selected test cases.

Having sufficiently trained their support vector machine, the authors then queried whether the model could predict synthesis methods for compounds not present in the training set, including the synthesis of two-dimensional zinc sulfide and cadmium sulfide. They noted that the support vector machine was able to accurately predict synthesis mechanisms for certain compounds, although it struggled on the synthesis of cadmium sulfide (a problem that the authors noted might be due to the relative youth of two-dimensional metal oxide synthesis in general).

Ultimately, this work suggests that machine learning (aided by other computational science techniques like natural language processing) could be used to aid in not only the theorizing of new materials with specific properties, but also in the ways in which those novel compounds could be synthesized.

3. Conclusion

As demonstrated, machine learning is capable of offering solutions to many problems plaguing the field of materials science today. These computational solutions often go beyond the capabilities of modern materials science methodologies, and thus the use of scientific computing within the field of materials science shows great promise for the future of both fields.

One particularly hard problem within materials science is the analysis of volumetric datasets, which can be rather difficult to do without some form of automation. To address this issue, one group of researchers used an unsupervised learning algorithm called density-based spatial clustering of applications with noise (DBSCAN) to visualize data gathered from neutron scanning tomography. They found that, compared to more typical reconstruction methods, DBSCAN produced a better visualization of the internal structures with significantly less noise. Thus, machine learning can be used to improve the analysis process, rather than just as a simple analytical tool.

Another way that machine learning has been used is as a complement to the capabilities of current methodologies in materials science. The hardness of a material cannot be readily predicted by current methods because hardness is a non-equilibrium property and density functional theory, which simulates materials using quantum chemistry, is not optimized to predict material behavior outside of equilibrium. This means that predicting which compounds will demonstrate superhard behavior is particularly difficult within conventional methods. To address this problem, researchers applied support vector regression to predict which

compounds out of a database of tens of thousands of compounds would be likely to demonstrate superhard behavior, and ultimately were able to identify several compounds that fulfilled that criterion when synthesized and tested experimentally. In this way, machine learning models can provide an alternative solution to problems where more conventional methods prove insufficient.

A third way that machine learning methods can be used is as a full replacement of existing techniques. Some currently ongoing work is evaluating the ability of generative adversarial networks to simulate grain growth and microstructural evolution in transition metals. Grain growth simulations can be immensely expensive in terms of computational time, on the order of hours of computational time for seconds of simulation time; in fact, they can be even more expensive than the training of convolutional neural networks in terms of complexity and computational time required. Preliminary results from this research suggest that, once trained, the network can accurately predict the evolution of grains in an arbitrary image faster than running that same image through existing material simulation technologies. The ultimate goal of this research would be to shift the primary computational time cost from a continuous cost of running expensive simulations to a more fixed, up-front cost of training a set of adversarial networks instead, and then using those trained networks to simulate grain growth over time.

The final application of machine learning discussed in this paper sought to address the relatively limited amount of materials science data. By compiling a database of thousands of articles about metal oxide synthesis, researchers were able to produce an artificial intelligence that could predict the synthesis methods of novel materials. This not only has the potential to vastly accelerate the synthesis of those materials, but also offers a way that the data within the field can be used to collectively advance science beyond the individual analysis or article for which it was produced. The ability to perform meta-analysis on specific subdomains of materials science is very promising for the purposes of advancing materials discovery and understanding.

Throughout this review, the application of machine learning to major materials science problems has been shown to offer immediate and effective solutions. Machine learning can be used to not only predict compounds with extraordinary properties, but also aid in the development of synthesis mechanisms for those materials. Machine learning can improve existing methods of analysis, and can even, in some cases, provide alternatives to current methods that may not be well-suited to specific applications. Machine learning can even address some of the largest problems in the field, such as the limited amount of available data and the importance of the ability to reuse not only experimental data but also information about other aspects of materials development, such as synthesis methods. To conclude, machine learning methodologies have a lot to offer the field of materials science, now and into the future.

References

de Albuquerque, V. H. C., Cortez, P. C., de Alexandria, A. R. & Tavares, J. M. R. S. (2008). A new solution for automatic microstructures analysis from images based on a backpropagation artificial neural network. *Nondestructive Testing and Evaluation*, *23*(4), 273–283. doi:10.1080/10589750802258986

Carrasquilla, J. & Melko, R. G. (2017). Machine learning phases of matter. *Nature Phys*, *13*, 431–434. doi:10.1038/nphys4035

Butler, K.T., Davies, D. W., Cartwright, H., Isayev, O. & Walsh, A. (2018). Machine learning for molecular and materials science. *Nature*, *59*, 547–555. doi:10.1038/s41586-018-0337-2

Hui, Y. & Liu, Y. (2018). Volumetric data exploration with machine learning-aided visualization in neutron science. arXiv:1710.05994

Ester, M., Kriegel, H., Sander, J. & Xu, X. (1996). A density-based algorithm for discovering clusters. *KDD Proceedings*, *96*, 226–231.

Geerlings, P., De Proft, F. & Langenaeker, W. (2003). Conceptual density functional theory. *Chem. Rev.*, *103*(5), 1793–1874. doi:10.1021/cr990029p

Zhao, Y., Ding, Z. & Zhou, S. (2004). Hardness and fracture toughness of brittle materials: a density functional theory study. *Phys. Rev. B*, *70*(18). doi:10.1103/PhysRevB.70.184117

Zhang, H. (2011). *Building Materials in Civil Engineering*. Woodhead.

Phaal, C. (1991). *United States patent No. US5007207A*.

Lux, B. & Haubner, R. (1994). Diamond as a wear-resistant coating. *Thin Film Diamond*. Springer. doi:10.1007/978-94-011-0725-9

Kaner, R. B., Gilman, J. J. & Tolbert, S. H. (2005). Designing superhard materials. *Science*, *308*(5726), 1268–1269. doi:10.1126/science.1109830

Tehrani, A. M., Ghadbeigi, L., Brgoch, J. & Sparks, T. D. (2017). Balancing mechanical properties and sustainability in the search for superhard materials. *Integrating Materials and Manufacturing Innovation 6*, 1–8. doi:10.1007/s40192-017-0085-4

Brgoch, J., Tehrani, A. M., Oliynyk, A. O., Parry, M., Rizvi, Z., Couper, S., Lin, F., Miyagi, L. & Sparks, T. D. (2018). Machine learning directed search for ultraincompressible, superhard materials. *J. Am. Chem. Soc. 140*(31) 9844–9853. doi:10.1021/jacs.8b02717

Liu, A. Y & Cohen, M. L. (1989). Preidction of new low compressibility solids. *Science*, *245*(4920), 841–842. doi:10.1126/science.245.4920.841

Kim, S. P. & Bush, M. B. (1999). The effects of grain size and porosity on the elastic modulus of nanocrystalline materials. *Nanostructed Materials*, *11*(3) 361–367. doi:10.1016/S0965-9773(99)00052-5

Badwal, S. P. S. & Drennan, J. (1987). Vyttria-zirconia: effect of microstructure on conductivity. *J. Materials Science*, *22*(9), 3231–3239. doi:10.1007/BF01161187

Santos, H. A., Riikonen, J., Salonen, J., Makila, E., Heikkila, T., Laaksonen, T., Peltonen, L., Lehto, V. & Hirvonen, J. (2010). In vitro cytotoxicity of porous silicion microparticles: effect of the particle concentration, surface chemistry and size. *Acta Biomaterialia*, *6*(7), 2721–2731. doi:10.1016/j.actbio.2009.12.043

Mohamed, F. A. & Langdon, T. G. (1974). Deformation mechanism maps based on grain size. *Metallurgical Transactions*, *5*, 2339–2345. doi:10.1007/BF02644014

Zaeem, M. A., Kadiri, H. E., Horstemeyer, M. F. & Wang, P. T. (2011). The roles of grain boundary energy anisotropy and second-phase particles on grain growth in polycrystalline materials. *Proceedings of IMETI*, *1*, 180–182.

Tang, M., Yang, K., Cao, Y., & Hitt, A. *Unpublished work*.

Hu, Z., Jujjavarapu, C., Hughey, J. J., Andorf, S., Lee, H. C., Gherardini, P. F., Spitzer, M. H., Thomas, C. G., Campbell, J., Dunn, P., Wiser, J., Kidd, B. A., Dudley, J. T., Nolan, G. P., Bhattacharya, S. & Butte, A. J. (2018). MetaCyto: a tool for automated meta-analysis of mass and flow cytometry data. *Cell Rep.*, *24*(5), 1377–1388. doi:10.1016/j.celrep.2018.07.003.

Olivetti, E., Kim, E., Huang, K., Saunders, A., McCallum, A. & Ceder, G. (2017). Materials synthesis insights from scientific literature via text extraction and machine learning. *Chem. Mater.*, *29*(21), 9436–9444. doi:10.1021/acs.chemmater.7b03500

Double Acceleration: Effects of Using Momentum-based Methods on Overparameterized Models

Mohammad Taha Toghani [1] **Delaram Pirhayatifard** [1]

Abstract

Overparameterization has been of considerable interest in deep learning recently. It challenges the general belief that increasing depth in a neural network simply leads to the complexity of the algorithm. Yet, by theoretical justification, it has also been proven that it boosts convergence rate significantly. In addition to the overparameterization, the same effects have been observed for the asynchronous distributed machine learning. In this article, we provide a literature review on the implicit acceleration methods including those connected with overparameterization and asynchrony. Then, we investigate the "Double Acceleration" phenomena where an optimization problem is being affected by both implicit and explicit accelerators and show that it changes the inherent value of momentum in optimization.

1. Introduction

In deep learning there has constantly been an attempt to accelerate the existing learning algorithms. Stochastic gradient descent revolutionized optimization for the large-scale machine learning at the very beginning, then second-order methods such as momentum acceleration introduced with a more amazing performance to the extent that most state-of-the-art models are trained using it. Where momentum is defined as the moving average of gradients. This method is an example of explicit acceleration. Yet, there are also implicit methods. Implicit acceleration through overparameterization (Arora et al., 2018b) has been of specific interest in the sense of demonstrating acceleration through increasing the parameters in the simple ℓ_p regression problem or equivalently depth in linear neural networks, where the trade-off between expressiveness and optimization is investigated. Another method has been proposed for implicit acceleration by (Mitliagkas et al., 2016) where they have demonstrated running stochastic gradient descent(SGD) asynchronously can lead to momentum-like behavior. In this work, by empirical evaluation as well as some theoretical justifications, we intend to suggest that applying two accelerators to the learning algorithm can make the optimization too much accelerated, where we use the momentum method instead of gradient descent. In (Arora et al., 2018a), the convergence rate of the momentum implied by overparameterization under a few assumptions has been discussed and a relaxed condition for initialization has been suggested in comparison to (Arora et al., 2018b). The remainder of the paper is organized as follows. In section 2 we investigate existing state-of-the-art methods for implicit acceleration. In section 3 applying an explicit momentum accelerator on an overparameterized case is discussed and we discuss the valid and optimal value of momentum along with theoretical investigations.

[1]Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA. Correspondence to: Mohammad Taha Toghani <mttoghani@rice.edu>, Delaram Pirhayatifard <dp43@rice.edu>.

2. Related Works

2.1. Asynchrony

In distributed machine learning, one of the important intentions is to train a common model using distributed parties. These parties are called workers and usually (but not necessarily) there is a central part, server, which all parties have access to. Here, assume that all workers want to optimize common parameters using an iterative optimization algorithm like SGD, while each one of them has access to a portion of the dataset. Thus, each worker computes the gradient over its dataset and sends it back to the server. If all parties work harmonious, this process is called synchronous and otherwise, it is asynchronous. Note that asynchronous methods are more efficient in terms of hardware so if are of great importance for practitioners in this field. So, one of the imperative subjects is to analyze their functionality in terms of optimization. (Mitliagkas et al., 2016) showed that asynchrony leads to an implicit acceleration similar to what momentum does. In the rest of this subsection, we are going to review their results.

2.1.1. Problem Description

Assume that the aim is to minimize the empirical risk of a model as follows:

$$f(\mathbf{w}) =: \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{w}, z_i) \tag{1}$$

where $f : \mathbb{R}^{d_x} \to \mathbb{R}$ and $f_i(.)$ is the cost function on a mini-batch of the whole data, like z_i. Using SGD, in each step, $\mathbf{w}$ must be updated as follows:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta_t \nabla_{\mathbf{w}} f(\mathbf{w}^{(t)}; z_{i_t}) \tag{2}$$

The above update rule belongs to a synchronous learning regime where in i_t represents which batch to be chosen, z_{i_t} and the worker who owns z_{i_t} takes care of this update. Also, its asynchronous counterpart is the same i.e. each worker reads the current value of common parameter, computes the gradient, and then updates it. Just the difference is that workers update the parameter carelessly of each other which induces a delay to the update. Formally, the update rule in the asynchronous regime is exactly same as Equation 2, but the gradient is $\nabla_{\mathbf{w}} f(\mathbf{v}^{(t)}; z_{i_t})$ where in $\mathbf{v}^{(t)} = \mathbf{w}^{(t-\tau_t)}$. Note that τ_t is the delay imposed by the asynchronous interaction of workers on the common parameter which is called staleness. To alleviate the complexity of the problem, assume that for each worker at time t, the delay follows distribution Q:

$$\mathbf{v}^{(t)} = \mathbf{w}^{(t-l)} \qquad w.p. \quad q_l, \quad l \in \mathbb{N} \tag{3}$$

thus, according to the above model on randomness of the delay, we can infer that:

$$\mathbb{E}[\mathbf{v}^{(t)}] = \sum_{l=0}^{\infty} q_l \, \mathbb{E}[\mathbf{w}^{(t-l)}] \tag{4}$$

which means the expectation of the read value is the convex combination of all values up to now. Also, assume that τ_t and i_t are independent which is a reasonable assumption for this problem.

2.1.2. Asynchrony Implicit Momentum

Theorem 1 *Using SGD with a constant step size $\eta_t = \eta$, the following result holds:*

$$\mathbb{E}[\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}] = \mathbb{E}[\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}] - \eta q_0 \, \mathbb{E}\nabla_{\mathbf{w}} f(\mathbf{w}^{(t)}) + \eta \sum_{l=0}^{\infty} (q_l - q_{l+1}) \, \mathbb{E}\nabla_{\mathbf{w}} f(\mathbf{w}^{(t-(l+1))}) \tag{5}$$

Corollary 2 *If Q is a geometric distribution such that $q_l = (1-\mu)\mu^l$, then:*

$$\mathbb{E}[\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}] = \mu \, \mathbb{E}[\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}] - (1-\mu)\eta \, \mathbb{E}\nabla_{\mathbf{w}} f(\mathbf{w}^{(t)}) \tag{6}$$

where, the momentum is μ.

Theorem 3 *Let's denote the time it takes to each worker to finish step t as W_t. If W_t is exponentially distributed with parameter λ ($W_t \sim Exp(\lambda)$) and there are M asynchronous workers, then:*

$$\mathbb{E}[\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}] = \left(1 - \tfrac{1}{M}\right) \mathbb{E}[\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}] - \tfrac{\eta}{M} \, \mathbb{E}\nabla_{\mathbf{w}} f(\mathbf{w}^{(t)}) \tag{7}$$

The exponential distribution assumption is a simple queuing model to elaborate the analysis

According to the result of Theorem 3, $\beta = \left(1 - \frac{1}{M}\right)$ is the imposed momentum by the asynchrony and M is called the degree of asynchrony. Now, assume that the optimal momentum to optimize a model is $\beta^\star$, so if we increase the number of workers such that $\beta^\star < \left(1 - \frac{1}{M}\right)$ then the implicit momentum of problem passes the optimal value. On the other hand, if M is not high enough, the implicit momentum would not be enough to get the optimal value, so adding an explicit momentum seems can compensate for the shortage of momentum. Fig. 1 shows this effect very well. In the next part, we discuss the interaction of both explicit and implicit momentum induced by asynchrony in detail.

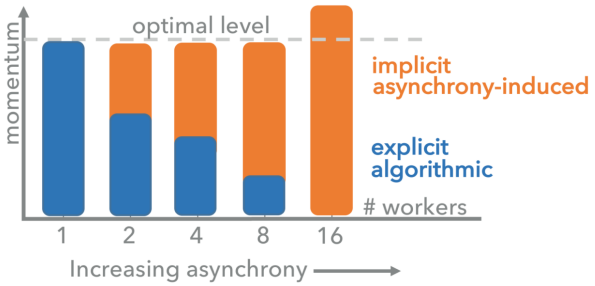


Figure 1. Borrowed from (Mitliagkas et al., 2016)-Total momentum has some optimal value. When implicit momentum is less than that, we can algorithmically compensate for the rest. Beyond a certain point, asynchrony causes too much momentum, leading to statistical inefficiency.

2.1.3. Explicit & Implicit Momentum

As we discussed in the last part, asynchrony induces an implicit momentum to SGD. Now the question is what if we add asynchrony to an explicit optimizer like Heavy-Ball momentum? The following Theorem shows a closed-form answer for the interaction of these two.

Theorem 4 *If each worker uses Heavy-Ball momentum algorithm to update common parameters as follows:*

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} f(\mathbf{w}^{(t-\tau_t)}) + \beta(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}) \tag{8}$$

and Q is like what is in Corollary 2, then we have:

$$\begin{aligned}\mathbb{E}[\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}] =& \big(\beta + \mu\big)\,\mathbb{E}[\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}] \\ &- \beta\mu\,\mathbb{E}[\mathbf{w}^{(t-1)} - \mathbf{w}^{(t-2)}] \\ &- (1-\mu)\eta\,\mathbb{E}\,\nabla_{\mathbf{w}} f(\mathbf{w}^{(t)})\end{aligned} \tag{9}$$

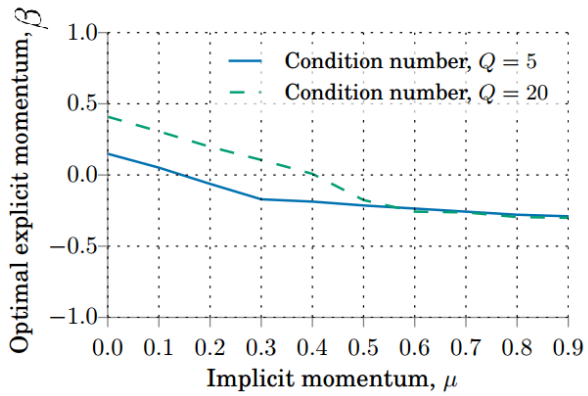


Figure 2. Borrowed from (Mitliagkas et al., 2016)-The optimal value of explicit momentum that causes the faster convergence for different values of implicit momentum. As the value of implicit momentum caused by staleness grows, the optimal value of explicit momentum decreases. For higher staleness, the optimal explicit momentum is negative.

Thus, conform to the result of Theorem 4, we intuitively expect that the intrinsic momentum of the problem will be increased proportionally to the summation of staleness (μ) and β. It's a convention that people in deep learning use $\beta = 0.9$ and generally $\beta \in [0, 1)$. But if the momentum induced by asynchrony is greater than the optimal momentum, then using a momentum-based algorithm like Heavy-Ball with a positive momentum decelerates the optimization. So, what if we use a negative β? Below, we state a theorem to compute the convergence rate for quadratic objectives and then we will discuss the optimal β which should be used in the Heavy-Ball method in order to achieve the best convergence.

Theorem 5 *(Borrowed from (Mitliagkas et al., 2016)) Assume that f is a simple quadratic objective like $f(\mathbf{w}) = \frac{1}{2}\|\mathbf{A}\mathbf{w} - \mathbf{b}\|_2^2$, where $\mathbf{A}\mathbf{w}^\star = \mathbf{b}$. Let λ_i denote the i-th eigenvalue of $\mathbf{A}^\top\mathbf{A}$ and $t_i^\star$ denote the root of smallest magnitude for the polynomial*

$$g_i(t) = \mu\beta t^3 - (\mu + \beta + \mu\beta)t^2 + z_i t - 1, \tag{10}$$

where

$$z_i = 1 + \mu + \beta - \eta(1-\mu)\lambda_i. \tag{11}$$

The convergence rate of the expected iterates in the statement of Theorem 4 is given by

$$\|\,\mathbb{E}\,\mathbf{w}_t - \mathbf{w}^\star\|_2 = O(\gamma^t), \tag{12}$$

where

$$\gamma = \max_i \frac{1}{|t_i^\star|}. \tag{13}$$

According to Theorem 5, we can compute the best convergence rate as a function of β, μ, η, and $\mathbf{A}$. Thus, for a simple quadratic model, for each fixed staleness we can find the optimal value of explicit momentum using a grid search on β and the learning rate. As shown in Fig. 2, for two quadratic models with $Q = 5, 20$, for high values of staleness, the optimal explicit momentum is negative. We can infer the double acceleration phenomena in the figure. So, the valid interval of β has been changed using an implicit momentum factor, i.e. it can be seen that somehow, there is an upper bound on the summation of both explicit and implicit momentum. In an extreme case like the very high value of implicit momentum caused by staleness, a negative explicit momentum can rectify the asynchrony harm.

2.2. Overparameterization

In machine learning, it is assumed that there is a trade-off between expressiveness and complexity. As a result of this belief, increasing depth leads to a more expressive model while it makes everything complex, including the optimization. (Arora et al., 2018b;a) have recently shown that adding the number of parameters implicitly accelerates the convergence speed. In fact, they have shown that under some assumptions, more parameters can induce an implicit momentum behavior. In the rest of this subsection, we first discuss a simple reparameterization model then review the case for linear neural networks. Finally, will discuss the speed of convergence to the global optimum in a linear neural network using SGD.

2.2.1. Simple Reparameterization

In order to demonstrate the significant effect of overparameterization on convergence, consider the simple regression problem with ℓ_p loss function:

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{(x,y)\sim S}[\frac{1}{p}(\mathbf{x}^T\mathbf{w} - y)^p] \tag{14}$$

Where $\mathbf{x} \in \mathbb{R}^{d_x}$, $\mathbf{w} \in \mathbb{R}^{d_x}$ and $y \in \mathbb{R}$. To find the optimal answer for the above cost function, we can use an iterative method like SGD (similar to what is mentioned in Equation 2) To observe the effect of adding parameters, let's reparameterize $\mathbf{w}$. The simple underlying idea is to write $\mathbf{w}$ as the product of a scalar and a vector, $\mathbf{w} = \mathbf{w_1}w_2$. By this assumption, in order to apply SGD, we use alternative gradient descent. In other words, the updating rule for the

parameters will be as follows:

$$\begin{aligned} \mathbf{w}_1^{(t+1)} &\leftarrow \mathbf{w}_1^{(t)} - \eta \nabla_{\mathbf{w}_1^{(t)}} \\ w_2^{(t+1)} &\leftarrow w_2^{(t)} - \eta \nabla_{w_2^{(t)}} \end{aligned} \tag{15}$$

It can be easily shown that $\nabla_{\mathbf{w}_1^{(t)}} = w_2^{(t)} \nabla_{\mathbf{w}^{(t)}}$, so we can then derive the update rule for $\mathbf{w}$ as follows:

$$\begin{aligned} \mathbf{w}^{(t+1)} =& \mathbf{w}_1^{(t+1)} w_2^{(t+1)} \\ =& \mathbf{w}^{(t)} - \eta (w_2^{(t)})^2 \big(\nabla_{\mathbf{w}^{(t)}}\big) \\ & - \eta (w_2^{(t)})^{-1} \big(\nabla_{w_2^{(t)}}\big) \mathbf{w}^{(t)} + \mathcal{O}(\eta^2) \end{aligned} \tag{16}$$

In the above expression, assume that η is small enough such that η^2 is negligible and we can remove it. Therefore, expression 16 would be simplified as follows:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)} \nabla_{\mathbf{w}^{(t)}} - \beta^{(t)} \mathbf{w}^{(t)} \tag{17}$$

In Equation 17, $\eta^{(t)}$ is a variable learning rate and $(-\beta^{(t)})$ is a variable momentum-like factor. Note that both $\eta^{(t)}$ and $(-\beta^{(t)})$ depend on the current point in SGD. So, it can be inferred that adding an extra parameter has implicitly led to a momentum-like behaviour. Fig. 3 has illustrated this phenomenon.

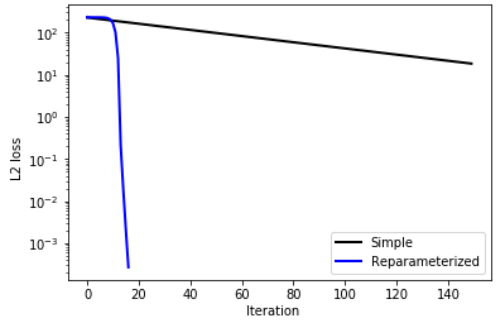


Figure 3. Comparison between simple and reparameterized models.

2.2.2. Linear Neural Networks Training

In this part we aim to generalize the idea of simple reparameterization. Let's assume that $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ such that $\mathcal{X} \subset \mathbb{R}^{d_x}$, $\mathcal{Y} \subset \mathbb{R}^{d_y}$. So if we want to predict y based-on x using regression. So there is a $\mathbf{W} \in \mathbb{R}^{d_y \times d_x}$ ($\mathbf{y} = \mathbf{W}\mathbf{x}$) and a loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ like ℓ_2-loss. The idea of is to overparameterize the model as follows:

$$\mathbf{W} = \mathbf{W_N} \mathbf{W_{N-1}} \dots \mathbf{W_1} \tag{18}$$

such that $\mathbf{W_j} \in \mathbb{R}^{n_j \times n_{j-1}}$ where $n_0 = d_x$ and $n_N = d_y$. The model in Equation 18 is called a linear neural network since it is exactly similar to a dense neural network, the only difference is that between every two linear operators there is a non-linear layer. Hereinafter for simplicity we denote the loss of a depth-N linear neural network as $L^N(\mathbf{W_1}, \mathbf{W_2}, \dots, \mathbf{W_N})$. So as a simple deduction of this definition:

$$L^1(\mathbf{W}) = L^N(\mathbf{W_1}, \mathbf{W_2}, \dots, \mathbf{W_N}) \tag{19}$$

Using the alternative gradient descent to find the optimal answer, we know that:

$$\begin{aligned} \mathbf{W}_j^{(t+1)} \leftarrow \mathbf{W}_j^{(t)} - \eta \frac{\partial L^N}{\partial \mathbf{W}_j} (\mathbf{W_1}^{(t)}, \dots, \mathbf{W_N}^{(t)}) \\ \forall j : \quad j \in [N] \end{aligned} \tag{20}$$

We intend to find a closed-form update rule based on $L^1(.)$ and $\mathbf{W}$, so a the idea in (Arora et al., 2018b) is to assume that t is continuous and derive the Equations based on that. Also note that η^2 is negligible. Let's denote the derivative of $\mathbf{W}_j(t)$ with respect to the time as $\dot{\mathbf{W}}_j(t)$, so the update rule in 20 in the continuous form is as follows:

$$\begin{aligned} \dot{\mathbf{W}}_j(t) = -\eta \frac{\partial L^N}{\partial \mathbf{W}_j} (\mathbf{W_1}(t), \dots, \mathbf{W_N}(t)) \\ \forall j : \quad j \in [N] \end{aligned} \tag{21}$$

2.2.3. Implicit Momentum

In this part, we want to show that with a proper initialization, the update rule in Equation 21 turns into a closed-form update rule with a momentum term in it.

Lemma 6 *Assume that a neural network is initialized as follows:*

$$\begin{aligned} \mathbf{W}_{j+1}^\top(t_0) \mathbf{W}_{j+1}(t_0) = \mathbf{W}_j(t_0) \mathbf{W}_j^\top(t_0) \\ \forall j : \quad j \in [N-1] \end{aligned} \tag{22}$$

(this initialization is called perfect initial balancedness) then for all j such that $j \in [N-1]$ and $\forall t$:

$$\mathbf{W}_{j+1}^\top(t) \mathbf{W}_{j+1}(t) = \mathbf{W}_j(t) \mathbf{W}_j^\top(t) \tag{23}$$

Theorem 7 *Assume that a depth-N linear neural network is being trained with SGD (Equation 21) and is initialized according to Equation 22, then the following differential equation holds:*

$$\begin{aligned} \dot{\mathbf{W}}(t) = -\eta \sum_{j=1}^{N} \Big[\mathbf{W}(t)\mathbf{W}^\top(t)\Big]^{\frac{j-1}{N}} \\ . \frac{d}{d\mathbf{W}} L^1(\mathbf{W}(t)) . \Big[\mathbf{W}(t)\mathbf{W}^\top(t)\Big]^{\frac{N-j}{N}} \end{aligned} \tag{24}$$

where $[.]^{\frac{j}{N}}$ is the fractional power operator over PSD matrices.

It's enough to substitute $\dot{\mathbf{W}}(t)$ with $\mathbf{W}^{(t+1)} - \mathbf{W}^{(t)}$, in Equation 7 to obtain the update rule for discrete t. This update rule relies on the fact that the learning rate is small enough and parameters are initialized properly. In neural network the initialization assumption is the case since all parameters are initialized near zero but specially in 2.2.4, we will introduce a new notation under which this assumption is milder.

Apparently, the term on the right-hand side of Equation 7, is not complied with non of the optimization algorithms. Claim 8 represents this update rule in an interpretable format.

Claim 8 *For an arbitrary matrix* $\mathbf{A}$*,* $vec(A)$ *denotes its arrangement as a vector in column-first order. The update rule of linear neural network can be written as:*

$$vec(\mathbf{W}^{(t+1)}) \leftarrow vec(\mathbf{W}^{(t)}) - \eta P_{\mathbf{W}^{(t)}} vec\left(\frac{d\,L^1}{d\,\mathbf{W}}\left(\mathbf{W}^{(t)}\right)\right) \tag{25}$$

where $P_{\mathbf{W}^{(t)}}$ *is a PSD preconditioning matrix based on* $\mathbf{W}^{(t)}$.

In the above claim, the preconditioning matrix modifies the gradient direction for update rule. In other words, (if we look at details) it can be seen that it modifies the movement direction such that it falls in line with the direction of current position of the algorithm ($\mathbf{W}^{(t)}$). This effect can be interpreted as a momentum-like behaviour in which the depth growth phenomena tries to alter the gradient in line with the current position of the algorithm. Note that, the momentum effect is not exactly vivid yet, however the next corollary, clarifies this effect brightly.

Corollary 9 *For* $d_y = 1$*, the update rule in Equation 8 can be written as follows:*

$$\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t)} - \eta \|\mathbf{W}^{(t)}\|_2^{(2-\frac{2}{N})}\left(\frac{d\,L^1}{d\,\mathbf{W}}\left(\mathbf{W}^{(t)}\right) + (N-1) Pr_{\mathbf{W}^{(t)}}\left\{\frac{d\,L^1}{d\,\mathbf{W}}\left(\mathbf{W}^{(t)}\right)\right\}\right) \tag{26}$$

where in $Pr_{\mathbf{W}^{(t)}}(.)$ *stands for the projection operator onto the direction of* $\mathbf{W}^{(t)}$.

Equation 26, reminds us Equation 17 since the generated parameters can be interpreted as a variable learning rate and an adaptive implicit momentum. In other words, regarding the fact that parameters are initialized near zero, $\mathbf{W}^{(t)}$ represents contains the history of all movements in the optimization process. As a result, when the optimization paves the way, the learning rate gets more confident and get adjusted. Moreover, the projected gradient along with the coefficient $N-1$ implies an adaptive momentum proportional to the depth and the correlation between the gradient and the history of movements. To see the effect of adding parameters in practice, we generate synthetic data from a multivariate linear model and try to learn parameters with SGD. Then similar to what we did in 2.2.1, we increase the number of parameters, i.e. $\mathbf{w} = \mathbf{w}_1 w_2 w_3 \dots w_i$ that is a depth-i model. Note that we chose η small enough such that none of the models diverge. Fig. 4-Right shows the potential effect of adding parameters to accelerate the algorithm. Fig. 4-Left is the result of the algorithm with random initialization. As you see adding the number of parameters does not necessarily work better. To attain the proper initialization, we initialized the diagonal of each parameter with 1 and the rest with 0. It can be clearly seen that for the case with $n_1 = n_2 = \dots = n_{(N-1)} = \min\{n_0, n_N\}$ this initialization leads to Equation 22.

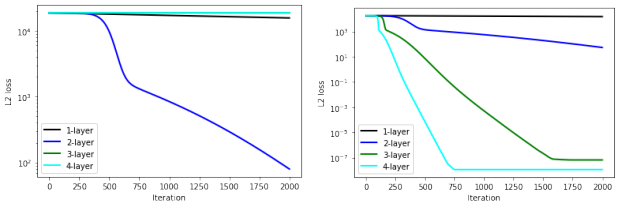


Figure 4. Left: Layers are initialized randomly near zero. Right: Layers are initialized according to Equation 22

2.2.4. Speed of Convergence

In the last part, we observed that adding the number of parameters substitute adds an inherent momentum to SGD but we did not discuss the possibilities for its convergence. The ultimate aim of (Arora et al., 2018a) is to show that a deep linear neural network by minimizing the ℓ_2 loss using SGD over a whitened data converges to a global minimum with a linear rate under the following conditions:

1. Dimensions of hidden layers are at least the minimum of the input and output dimensions.
2. Weight matrices at initialization are approximately balanced.
3. The initial loss is smaller than the loss of any rank-deficient solution.

According to the setting we discussed in 2.2.2, assume that $\mathbf{X} \in \mathbb{R}^{d_x \times m}$ where m is the number of samples and each column represents a $\mathbf{x} \in \mathbb{R}^{d_x}$ new sample. Similarly, $\mathbf{Y} \in \mathbb{R}^{d_y \times m}$ and $\mathbf{W} \in \mathbb{R}^{d_y \times d_x}$. Then the formal optimization problem is:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{2m}\|\mathbf{W}\mathbf{X} - \mathbf{Y}\|_F^2 \tag{27}$$

The assumption here is that X is whitened i.e. $\frac{1}{m}\mathbf{X}\mathbf{X}^\top = \mathbf{I}$ (the empirical uncentered covariance for instances is identity), so it can be shown that the above equation turns to:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{2}\|\mathbf{W} - \mathbf{\Phi}\|_F^2 + c \tag{28}$$

where $\mathbf{\Phi} = \frac{1}{m}\mathbf{Y}\mathbf{X}^\top$ and c is independent of $\mathbf{W}$. Thus, we should minimize the first expression in the right-hand side of Equation 28. Hereinafter, using N different matrix and substituting $\mathbf{W}$ with their product ($\mathbf{W} := \mathbf{W}_{1:N} = \mathbf{W}_1\mathbf{W}_2\dots\mathbf{W}_N$) we overparameterize the model and investigate how to satisfy the 3 aforementioned conditions.

Definition 10 *For $\delta \geq 0$, we say that the matrices $\mathbf{W}_j \in \mathbb{R}^{d_j \times d_{j-1}}$, $j \in [N]$, are δ-balanced if:*

$$\|\mathbf{W}_{j+1}^\top\mathbf{W}_{j+1} - \mathbf{W}_j\mathbf{W}_j^\top\|_F \leq \delta \tag{29}$$

If $\delta = 0$, then it implies the special initialization in Equation 22. The approximated balancedness is more useful than the perfect balancedness owing to the fact that layers are usually being initialized by random Gaussian with mean zero which does not lead to a perfect balancedness. It can be shown that if the weights of a linear neural network are initialized to be δ-balanced, then they will stay the same during the iterations of gradient descent.

Definition 11 *Given a target matrix $\mathbf{\Phi} \in \mathbb{R}^{d_N \times d_0}$ and a constant $c > 0$, we say that a matrix $\mathbf{W}$ of same size as $\mathbf{\Phi}$ has deficiency margin c with respect to $\mathbf{\Phi}$ if:*

$$\|\mathbf{W} - \mathbf{\Phi}\|_F \leq \sigma_{min}(\mathbf{\Phi}) - c \tag{30}$$

where $\sigma_{min}(\mathbf{\Phi})$ represents the smallest singular value of $\mathbf{\Phi}$.

The deficiency margin determines a ball with radius c around the target in which there is no rank-deficient matrix.

Corollary 12 *If $\mathbf{W}$ has deficiency margin c with respect to $\mathbf{\Phi}$, then any $\mathbf{W}'$ for which $\|\mathbf{W}' - \mathbf{\Phi}\|_F \leq \|\mathbf{W} - \mathbf{\Phi}\|_F$ satisfies $\sigma_{min}(\mathbf{W}') \geq c$.*

Let's denote $\ell(t)$ the loss in the t-th iteration:

$$\begin{aligned}\ell(t) = \mathcal{L}^1(\mathbf{W}^{(t)}) = \mathcal{L}^1(\mathbf{W}_{1:N}^{(t)}) = \\ = \mathcal{L}^N(\mathbf{W}_1^{(t)}, \dots, \mathbf{W}_N^{(t)})\end{aligned} \tag{31}$$

Theorem 13 *Assume that gradient descent is initialized such that the end-to-end matrix $\mathbf{W}_{1:N}^{(0)}$ has deficiency margin $c > 0$ with respect to the target $\mathbf{\Phi}$ and $\mathbf{W}_j^{(0)}$, $j \in [n]$ are δ-balanced with $\delta = \frac{c^2}{256.N^3.\|\mathbf{\Phi}\|_F^{2(N-1)/N}}$. Also, assume that $\eta \leq \frac{c^{(4N-2)/N}}{6144.N^3.\|\mathbf{\Phi}\|_F^{(6N-4)/N}}$. Then, for any $\epsilon > 0$:*

$$\ell(T) \leq \epsilon \tag{32}$$

where

$$T \geq \frac{1}{\eta.c^{2(N-1)/N}}.\log(\frac{\ell(0)}{\epsilon}) \tag{33}$$

According to the above theorem, we can infer that if the initialization satisfies the approximate balancedness condition for each matrix and simultaneously forces the end-to-end matrix to have a margin deficiency with respect to the target, then SGD converges with a linear rate. Note that a high margin leads to a faster convergence according to Equation 33. In (Arora et al., 2018a), it has been shown that for a single output model ($d_y = 1$), random zero-centered Gaussian initialization with lower variance causes a deficiency margin with higher probability (if it is small enough the probability converges to 0.5). Moreover, we know that for a small variance, $\mathbf{W}_j$s are likely to be approximately balanced, while a non-negligible variance is required to have a greater c. Hence, there is a trade-off on the Gaussian variance value i.e. higher values may decrease the possibility of deficiency margin and violate the approximate balancedness chance but in the case deficiency margin condition satisfies, it has a greater c. Results in (Arora et al., 2018a) generalize those in (Bartlett et al., 2019), in the sense that if all layers have the same size and we initialize them with identity matrix then it can easily be seen that it satisfies the aforementioned conditions.

3. Double Acceleration

As we discussed in Subsection 2.1, asynchrony leads to an implicit momentum. Also, in Subsection 2.2 we observed the same effects for adding parameters. Now, the idea is to investigate whether there is any interaction between implicit acceleration yielded from overparameterization like what we talked about in 2.1.3 for asynchrony. Thus, in this section, we are going to investigate a challenging but novel case, where we apply explicit momentum acceleration along with overparameterization. What one can expect at the very first is to see that the intrinsic momentum changes in line with adding more parameters.

3.1. Optimizing Simple Reparameterized Model By Heavy-Ball Algorithm

Here, we do the analysis similar to 2.2.1 for the Heavy-Ball method, to see how does the combination of momentum and reparameterization acts. So, let's start with the alternative update rule:

$$\begin{aligned}\mathbf{w}_1^{(t+1)} =& \mathbf{w}_1^{(t)} - \eta\nabla_{\mathbf{w}_1^{(t)}} + \beta\Big(\mathbf{w}_1^{(t)} - \mathbf{w}_1^{(t-1)}\Big) \\ w_2^{(t+1)} =& w_2^{(t)} - \eta\nabla_{w_2^{(t)}} + \beta\Big(w_2^{(t)} - w_2^{(t-1)}\Big)\end{aligned} \tag{34}$$

so, let's compute the update rule for $\mathbf{w}$:

$$\begin{aligned}
\mathbf{w}^{(t+1)} =& \mathbf{w}_1^{(t+1)} w_2^{(t+1)} && (35)\\
=& \left(\mathbf{w}_1^{(t)} - \eta \nabla_{\mathbf{w}_1^{(t)}}\right)\left(w_2^{(t)} - \eta \nabla_{w_2^{(t)}}\right) && (36)\\
&+\beta\left(\mathbf{w}_1^{(t)} - \eta \nabla_{\mathbf{w}_1^{(t)}}\right)\left(w_2^{(t)} - w_2^{(t-1)}\right) && (37)\\
&+\beta\left(\mathbf{w}_1^{(t)} - \mathbf{w}_1^{(t-1)}\right)\left(w_2^{(t)} - \eta \nabla_{w_2^{(t)}}\right) && (38)\\
&+\beta^2\left(\mathbf{w}_1^{(t)} - \mathbf{w}_1^{(t-1)}\right)\left(w_2^{(t)} - w_2^{(t-1)}\right) && (39)
\end{aligned}$$

we know that:

$$\begin{aligned}
(36) &= \mathbf{w}_1^{(t)} w_2^{(t)} - \eta w_2^{(t)} \nabla_{\mathbf{w}_1^{(t)}} - \eta \mathbf{w}_1^{(t)} \nabla_{w_2^{(t)}} + \mathcal{O}(\eta^2)\\
&= \mathbf{w}^{(t)} - \eta \big(w_2^{(t)}\big)^2 \nabla_{\mathbf{w}^{(t)}}\\
&- \left(\eta \big(w_2^{(t)}\big)^{-1} \nabla_{w_2^{(t)}}\right)\mathbf{w}^{(t)} + \mathcal{O}(\eta^2)
\end{aligned} \tag{40}$$

$$\begin{aligned}
(37) &= \mathbf{w}_1^{(t)} w_2^{(t)} - \mathbf{w}_1^{(t)} w_2^{(t-1)} - \eta w_2^{(t)} \nabla_{\mathbf{w}_1^{(t)}}\\
&+ \eta w_2^{(t-1)} \nabla_{\mathbf{w}_1^{(t)}}\\
&= \mathbf{w}^{(t)} - \mathbf{w}_1^{(t)} w_2^{(t-1)} - \eta \big(w_2^{(t)}\big)^2 \nabla_{\mathbf{w}^{(t)}}\\
&+ \eta w_2^{(t-1)} w_2^{(t)} \nabla_{\mathbf{w}^{(t)}}
\end{aligned} \tag{41}$$

$$\begin{aligned}
(38) &= \mathbf{w}_1^{(t)} w_2^{(t)} - \mathbf{w}_1^{(t-1)} w_2^{(t)} - \eta \mathbf{w}_1^{(t)} \nabla_{w_2^{(t)}}\\
&+ \eta \mathbf{w}_1^{(t-1)} \nabla_{w_2^{(t)}}\\
&= \mathbf{w}^{(t)} - \mathbf{w}_1^{(t-1)} w_2^{(t)} - \left(\eta \big(w_2^{(t)}\big)^{-1} \nabla_{w_2^{(t)}}\right)\mathbf{w}^{(t)}\\
&+ \left(\eta \big(w_2^{(t-1)}\big)^{-1} \nabla_{w_2^{(t)}}\right)\mathbf{w}^{(t-1)}
\end{aligned} \tag{42}$$

$$\begin{aligned}
(39) &= \mathbf{w}_1^{(t)} w_2^{(t)} - \mathbf{w}_1^{(t-1)} w_2^{(t)} - \mathbf{w}_1^{(t)} w_2^{(t-1)}\\
&+ \mathbf{w}_1^{(t-1)} w_2^{(t-1)}\\
&= \mathbf{w}^{(t)} - \mathbf{w}_1^{(t-1)} w_2^{(t)} - \mathbf{w}_1^{(t)} w_2^{(t-1)} + \mathbf{w}^{(t-1)}
\end{aligned} \tag{43}$$

based on the above equations, $\mathbf{w}^{(t+1)}$ can be written as follows:

$$\begin{aligned}
\mathbf{w}^{(t+1)} =& \mathbf{w}_1^{(t+1)} w_2^{(t+1)}\\
=& \mathbf{w}^{(t)} - \eta w_2^{(t)} \left((1+\beta) w_2^{(t)} - \beta w_2^{(t-1)}\right) \nabla_{\mathbf{w}^{(t)}}\\
&+ \left(2\beta + \beta^2 - \eta(1+\beta)\big(w_2^{(t)}\big)^{-1} \nabla_{w_2^{(t)}}\right)\mathbf{w}^{(t)}\\
&+ \left(\eta\beta \big(w_2^{(t-1)}\big)^{-1} \nabla_{w_2^{(t)}} + \beta^2\right)\mathbf{w}^{(t-1)}\\
&- \beta(\beta+1)\left(\mathbf{w}_1^{(t)} w_2^{(t-1)} + \mathbf{w}_1^{(t-1)} w_2^{(t)}\right)
\end{aligned} \tag{44}$$

To simplify the above result, let's use the following ideas:

- Idea 1: Let's choose η such that:

$$\begin{aligned}
\beta' &= 2\beta + \beta^2 - \eta(1+\beta)\big(w_2^{(t)}\big)^{-1} \nabla_{w_2^{(t)}}\\
&= -\eta\beta \big(w_2^{(t-1)}\big)^{-1} \nabla_{w_2^{(t)}} - \beta^2 \Rightarrow
\end{aligned} \tag{45}$$

$$\eta = \frac{2\beta(1+\beta)}{\left(\frac{(1+\beta)}{w_2^{(t)}} - \frac{\beta}{w_2^{(t-1)}}\right)\nabla_{w_2^{(t)}}} \tag{46}$$

  We can derive the final update rule for our case as follows:

$$\begin{aligned}
\mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} + \beta'\left(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}\right) - \eta' \nabla_{\mathbf{w}^{(t)}}\\
&- \beta(\beta+1)\left(\mathbf{w}_1^{(t)} w_2^{(t-1)} + \mathbf{w}_1^{(t-1)} w_2^{(t)}\right)
\end{aligned} \tag{47}$$

  where:

$$\eta' = \frac{2\beta(1+\beta) w_2^{(t)} \big((1+\beta) w_2^{(t)} - \beta w_2^{(t-1)}\big)}{\left(\frac{(1+\beta)}{w_2^{(t)}} - \frac{\beta}{w_2^{(t-1)}}\right)\nabla_{w_2^{(t)}}} \tag{48}$$

  According to Equation 47, the update rule for the product parameter is similar to Heavy-Ball with an additional term. The additional term seems to result in an extra momentum which is the reason for double acceleration.

- Idea 2: We know that:

$$\begin{aligned}
(\mathbf{w}_1^{(t)} - \mathbf{w}_1^{(t-1)}) &= \beta(\mathbf{w}_1^{(t-1)} - \mathbf{w}_1^{(t-2)}) - \eta \nabla_{\mathbf{w}_1^{(t)}}\\
&= \dots\\
&= -\eta \sum_{i=0}^{t} \beta^{(t-i)} \nabla_{\mathbf{w}_1^{(i)}}
\end{aligned} \tag{49}$$

similarly:

$$(w_2^{(t)} - w_2^{(t-1)}) = -\eta \sum_{i=0}^{t} \beta^{(t-i)} \nabla_{w_2^{(i)}} \tag{50}$$

let's simplify $\mathbf{w}_1^{(t)} w_2^{(t-1)} + \mathbf{w}_1^{(t-1)} w_2^{(t)}$, using Equations 49 & 50, thus:

$$\begin{aligned}(\mathbf{w}_1^{(t)} - \mathbf{w}_1^{(t-1)})(w_2^{(t)} - w_2^{(t-1)}) = \mathcal{O}(\eta^2) \Rightarrow \\ \mathbf{w}_1^{(t)} w_2^{(t-1)} + \mathbf{w}_1^{(t-1)} w_2^{(t)} = \\ \mathbf{w}_1^{(t)} w_2^{(t)} + \mathbf{w}_1^{(t-1)} w_2^{(t-1)} = \mathbf{w}^{(t)} + \mathbf{w}^{(t-1)}\end{aligned} \tag{51}$$

using the above result, we can re-write Equation 44 as follows:

$$\begin{aligned}\mathbf{w}^{(t+1)} =& \mathbf{w}^{(t)} - \eta w_2^{(t)} \Big((1+\beta) w_2^{(t)} - \beta w_2^{(t-1)} \Big) \nabla_{\mathbf{w}^{(t)}} \\ &+ \left(\beta - \eta \frac{(1+\beta)}{w_2^{(t)}} \nabla_{w_2^{(t)}} \right) \mathbf{w}^{(t)} \\ &+ \left(\eta \frac{\beta}{w_2^{(t-1)}} \nabla_{w_2^{(t)}} - \beta \right) \mathbf{w}^{(t-1)} \\ =& \mathbf{w}^{(t)} - \eta w_2^{(t)} \Big((1+\beta) w_2^{(t)} - \beta w_2^{(t-1)} \Big) \nabla_{\mathbf{w}^{(t)}} \\ &+ \left(\beta - \eta \frac{\frac{\beta}{w_2^{(t-1)}} + \frac{(1+\beta)}{w_2^{(t)}}}{2} \nabla_{w_2^{(t)}} \right) \\ &. \big(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)} \big) \\ &+ \eta \frac{\frac{\beta}{w_2^{(t-1)}} - \frac{(1+\beta)}{w_2^{(t)}}}{2} \nabla_{w_2^{(t)}} \big(\mathbf{w}^{(t)} + \mathbf{w}^{(t-1)} \big) \\ =& \mathbf{w}^{(t)} - \eta' \nabla_{\mathbf{w}^{(t)}} + \beta' \big(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)} \big) \\ &+ \alpha \big(\mathbf{w}^{(t)} + \mathbf{w}^{(t-1)} \big)\end{aligned} \tag{52}$$

According to Equation 52, applying the Heavy-Ball algorithm with β on reparameterized model leads to a momentum with a variable parameter, β', along with an additional term. The difference in the momentum term justifies the difference in the optimal momentum value that we talk about in the next Subsection.

3.2. Momentum Interval

To observe the effect of double acceleration, assume a simple quadratic objective like what we examined in Theorem 5. That being the case, we generate a synthetic data and finding the optimal value of $\mathbf{w}$, we reparameterize it as we discussed in 3.1 and use the Heavy-Ball method with different range for β. Simultaneously, we do the experiment on the non-reparameterized case and to have a reference for comparison, we apply these two simulations with SGD instead of Heavy-Ball.

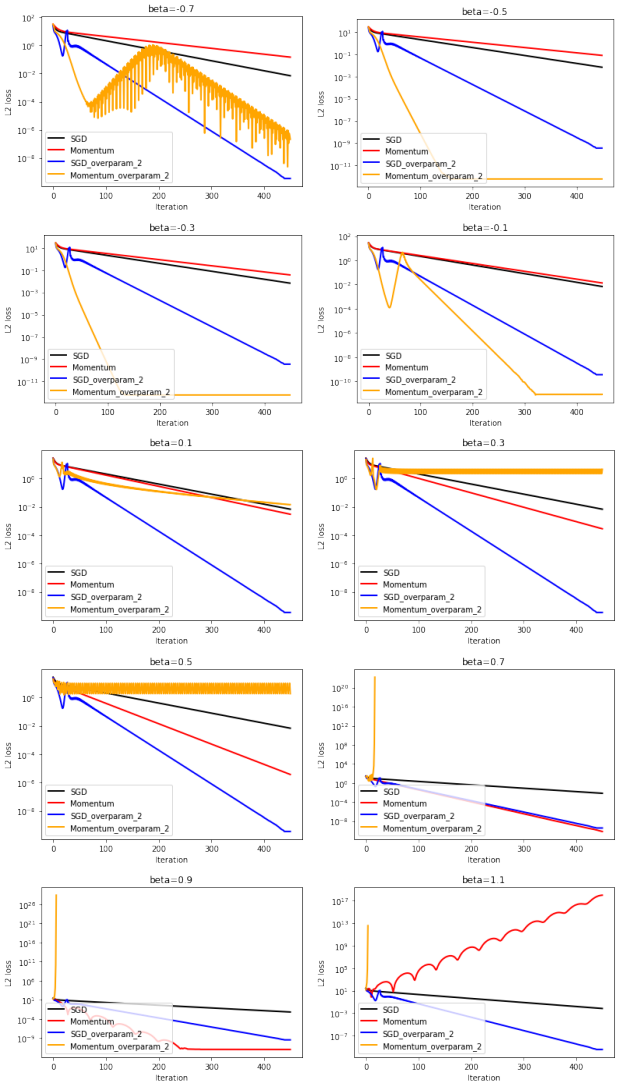


Figure 5. Effect of changing momentum value on convergence rate of different schemes.

Complied with Equation 52, we expect the range of β for the non-reparameterized case differs compared with its reparameterized counterpart. We optimize the non-reparameterized and reparameterized models using the SGD and Heavy-Ball method with different values of β from -0.7 to 1.1. As shown in Fig. 5, the optimal value of momentum for non-reparameterized model is about 0.9 and it seems the momentum method works better than SGD for $0 \leq \beta < 1$. However, for the reparameterized model, the optimal value of momentum is $\beta = -0.3$. Also, it seems the interval in which the Heavy-Ball optimizer surpasses SGD is $[-0.5, 0]$. We also observed a similar effect in Fig. 2. The resemblance between our results and those in (Mitliagkas et al., 2016) emphasizes the existence of the double acceleration phenomena wherein exploiting both implicit and explicit momentum leads to a higher value of momentum.

If the optimal momentum value for the non-reparameterized case is $\beta^\star$, then using this momentum value for a reparameterized case can lead to divergence. Additionally, we can infer that based on some circumstances, we may need to choose a non-routine value for momentum (for example a negative value) to obtain a proper convergence.

4. Conclusion

In this article we first did a literature review on the state-of-the-art notions of implicit acceleration, inspired by them, we continued by investigating a different case, double acceleration, that stands for exploiting both implicit and explicit accelerators. We examined the effect of using different values of hyper-parameters in convergence. One may expect that a powerful algorithm should be robust to the initialization, however theoretical analyses concede that it has a significant effect on convergence specifically in iterative methods such as SGD or momentum it somehow controls the trajectory of optimization. Then, in order to find the optimal momentum, we tried different values by empirical analysis. Note that implicit acceleration changes the intrinsic momentum of the algorithm, meaning the range of the appropriate explicit momentum (β) that leads to convergence is likely to change noticeably as we demonstrated that a negative momentum led to convergence while the popular choice for momentum, 0.9 diverged. Accordingly, the inherent momentum value that we derived is a combination of the explicit momentum and some other terms that causes a change in the value of momentum and then alters convergence.

References

Arora, S., Cohen, N., Golowich, N., and Hu, W. A convergence analysis of gradient descent for deep linear neural networks. *arXiv preprint arXiv:1810.02281*, 2018a.

Arora, S., Cohen, N., and Hazan, E. On the optimization of deep networks: Implicit acceleration by overparameterization. *arXiv preprint arXiv:1802.06509*, 2018b.

Bartlett, P. L., Helmbold, D. P., and Long, P. M. Gradient descent with identity initialization efficiently learns positive-definite linear transformations by deep residual networks. *Neural computation*, 31(3):477–502, 2019.

Mitliagkas, I., Zhang, C., Hadjis, S., and Ré, C. Asynchrony begets momentum, with an application to deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 997–1004. IEEE, 2016.

Exploring Label Smoothing Regularisation as an alternative to Knowledge Distillation

Aditya Desai Arindam Chowdhury

Abstract

Knowledge distillation has been successfully applied to model compression. Through a modified training objective, essential information captured by a deep network with complex architecture can be distilled into a shallow network, which significantly improves its overall accuracy at the same convergence rate and also adds to the generalization capacity. In spite of the popularity of this technique, there is a lack of fundamental theoretical understanding of its inner dynamics. There have been attempts to describe knowledge distillation as a form of regularization and attribute the strong performance gains to its ability to keep the model confidence in check by transferring class-similarity information which prevents overfitting. In this work, we investigate if this regularization effect can be enforced directly on a shallow network without assistance from a deep cumbersome network, thereby making the training process much simpler and efficient.

1. Introduction

Since its inception, deep neural networks have been applied to a multitude of learning tasks. These networks have enjoyed tremendous success in almost all modalities viz. images, text, video, speech, among many others. As problems are becoming more complex, the complexity of deep networks are growing exponentially in tandem with growth in high performance computational resources. Nevertheless, training these networks and storing the trained models is a difficult task, especially for online and incremental applications. Also, there is an increasing demand for algorithms that can be deployed on hand-held and low power devices, for example wearables for augmented and virtual reality. This essentially requires models to be computationally inexpensive and have smaller memory footprint while at the same time maintain reasonable performance on standard tasks.

Four broad categories of techniques have been proposed for model compression. Parameter pruning and sharing (Li et al., 2016) which involves reducing redundant parameters which are not sensitive to the performance. Low-rank factorization (Nakkiran et al., 2015) that uses matrix/tensor decomposition to estimate the informative parameters. Designing compact convolutional filters (Cohen & Welling, 2016) to save parameters and finally, knowledge distillation (Hinton et al., 2015) that involves training a compact neural network with distilled knowledge of a large model.

Among all these techniques, knowledge distillation is especially of interest as it involves transfer of *dark knowledge* from a trained deep network to a shallow network. Dark knowledge comprises of similarity information that is useful in learning correlations between true class and the rest. This effectively helps the shallow network to learn a label distribution essential for the task at hand, significantly improving is performance. One limitation of this method is that it requires a deep network to be trained first on data followed by training of the shallow network from its logits. This 2-step training process can be computationally expensive and also an hindrance to on-device training. Leveraging the inherent similarity of knowledge distillation to a standard regularization scheme, known as label smoothing regularization, we show that it can be formulated such that the requirement of a deep network is precluded and training process becomes simpler. In studying such a mechanism, we also gain important insights into the internal dynamics of the process and develop an understanding of its applicability in different settings.

This report is organized as follows. Section 2 introduces the concept of *Mimicking* that acts a precursor to *Knowledge Distillation*, which we formally present in Section 3. Section 4 describes a key limitation while suggesting an improvement over the vanilla version. *Label Smoothing Regularization* is introduced in Section 5 . We combine all the key ideas in Section 6 to establish a relationship between knowledge distillation and label smoothing regularization. In Section 7 we present and discuss some experimental results. We analyze knowledge distillation from yet another perspective in Section 8 which offers some interesting insights concluding our critical review.

2. Mimic Learning

Mimicking as a form of model compression was first proposed by (Ba & Caruana, 2013) who investigated the possibility of employing a shallow network to exact the performance of a relatively deeper network. Their main aim was to understand the effect of network depth on model performance, and in process answer the question *Do deep networks really need to be deep?*. They empirically demonstrate that shallow feed-forward networks can learn complex functions previously learned by deep nets and achieve accuracy previously only achievable by the later, if trained to approximate the learnt function instead of being trained on data. In fact, their experiments suggest that a model directly trained on the original data performs worse than a model trained with the help of a deeper model.

The SNN-MIMIC objective is formulated as a regression problem that involves minimization of $L2$-distance between the output *logits* of a shallow network being trained and their counterparts from a deep network pre-trained with *cross entropy* loss. Experiments with SNN-MIMIC show encouraging results on multiple classification tasks. For example, on CIFAR-10 dataset, a shallow network having 1 convolutional and 1 pooling layer with a final linear mapping, achieves a gain of $\approx 7\%$ when trained with a CNN ensemble as compared to training directly on data.

Authors explain the performance gain in terms of relational information between classes captured in the logits. Essentially, training on these logits make learning easier for the shallow net as, unlike in cross-entropy objective where only the true class probability is maximized, it has to learn the relative importance of each class in classifying an example. Moreover, the mean square objective ensures that the logits capture the logarithm relationships between the probability predictions. A shallow *student* model trained on logits has to learn all of the additional fine detailed relationships between labels that is not obvious in the probability space yet learned by the deep *teacher* model. By training the student model on the logits directly, the student is better able to learn the internal model learned by the teacher, without suffering from the information loss that occurs after passing through the logits to probability space.

Such a learning mechanism offers several additional advantages. Firstly, the teacher is able to eliminate some of the noise in training labels, thus allowing the student to learn from the information strictly relevant to the task at hand. Moreover, the original hard 0/1 labels can be difficult to learn given the features, sample density, and function complexity. In such cases, the teacher may provide simpler but more informative soft labels to the student. Also, it is often ineffective to learn from actual hard labels, as the entire probability mass is concentrated on a single class, without offering any correlation between training classes. This almost always leads to over-fitting. The model compressing technique discussed so far, provides a regularizing effect allowing better generalization performance of shallow networks.

This work established that it was possible, *in-principle*, for a shallow network to mimic the performance of a deep network without an increase in parameters. However, the algorithm for doing so was not well defined as it offered no control over how much of teacher's knowledge was relevant for the student and therefore strictly required the teacher model to be perfectly trained. In the next section, we discuss a more generalized approach that addressed this drawback while formalizing the student-teacher learning paradigm and introduced the very popular *knowledge distillation* framework.

3. Knowledge Distillation

As discussed in the last section, a lot of information is encoded in the relative importance of all other classes with respect to the true class. For example, in CIFAR dataset, the cat and dog classes are probably mapped much closer in a latent discriminative space and quite far from a bus and a train which themselves are mapped closer. Therefore, a possible hierarchy can be imagined, where similar objects group together and further away from vastly dissimilar objects. Thus, the correlation between classes makes the classification task easier. Entropy objective does not allow utilization of this information as it pushes the *softmax* prediction towards a one-hot structure, making the target distributions mutually orthogonal. As shown, one way to circumvent this problem is by using the logits, rather than the probabilities produced by the softmax, as the targets for learning the small model. A more general solution, called *distillation* was proposed by (Hinton et al., 2015), which raises the temperature T of the final softmax until the deep teacher model produces a suitably soft set of targets. The same high temperature is then used to train the shallow student model to match these soft targets. Temperature T is treated as a hyper-parameter, thereby allowing a control over the *softness* of the targets, lacking in the previous method. The modified softmax has the form

$$q_i(T) = \frac{exp(z_i/T)}{\Sigma exp(z_j/T)}$$

where z_i is i-th output logit. A combined objective function is then defined as

$$L = \alpha CE(q(1), y) + (1 - \alpha)T^2 CE(q(T), p(T))$$

where CE is the cross-entropy loss, y is true label and p is

the output distribution of the teacher. α is a hyper-parameter that controls a trade-off between the two loss components.

The use of an explicit cross-entropy loss, in addition to the matching loss, precludes the requirement of a perfectly trained teacher, as the model is also forced to predict the correct label while learning the relationship of the same with the incorrect labels. Nevertheless, best model performance is achieved for small values of α, i.e. when maximum focus is on matching the soft probabilities.

This formulation may look different from the logit matching algorithm under mimic learning framework but the authors show that matching the logits of the two models is actually a special case of distillation. In fact, at high temperatures, optimizing this loss is equivalent to optimizing MSE on the logits before softmax layer. Cross-entropy gradient for each logit z_i is given by $\frac{dC}{dz_i}$. Assuming, v_i to be teacher logits,

$$\frac{dC}{dz_i} = \frac{1}{T}\left(\frac{exp(z_i/T)}{\Sigma(exp(z_j/T))} - \frac{exp(v_i/T)}{\Sigma(exp(v_j/T))}\right)$$

For hight temperature, it can be approximated as

$$\frac{dC}{dz_i} \approx \frac{1}{T}\left(\frac{1 + z_i/T}{N + \Sigma_j z_j/T} - \frac{1 + v_i/T}{N + \Sigma_j v_j/T}\right)$$

Assuming the logits are zero-meaned separately, i.e. $\Sigma_j z_j = \Sigma_j v_j = 0$,

$$\frac{dC}{dz_i} \approx \frac{1}{NT^2}(z_i - v_i)$$

The gradient thus takes the form of gradient of mean square distance between z and v.

At lower values of T, distillation ignores logits that are strongly negative. This is advantageous as some of these logits could be highly noisy, thereby destabilizing training. Having said that, there is a possibility that the very negative logits may convey useful information about the knowledge acquired by the teacher. Authors show that for much shallower students who fail to capture all of the knowledge in the deeper model, intermediate temperatures work best. Thus it seems that ignoring the large negative logits can be helpful.

Intuitively, as the soft targets have higher entropy, they provide much more information per training case than hard targets and much less variance in the gradient between training cases. Therefore, a small network can learn from much less data than the original deep model and also converge faster than the deep model. Their experiments on MNIST and other domains also showed how effectively a model trained by distillation can learn to generalize. For example, even after removing all the instances of class 3 from the distillation training set, the smaller model mis-classifies examples of class 3 in the test set only $133/1010$ times, at an average. Fixing the bias of class 3 further reduces the error to a mere 14 out of 1000 test samples.

Since the introduction of this vital concept, several modifications have been proposed to address some of its fundamental drawbacks and to make it more efficient. In the next section we discuss some key concepts that provide deep insights into the dynamics of student-teacher relationship.

4. Improved Knowledge Distillation via Teacher Assistant

Our discussion so far has shown that knowledge distillation can be a promising way to obtain a shallow student model which retains the accuracy of a large teacher. This potentially opens up several possibilities. What if we can use the most accurate teacher to train the smallest student. In this section, we will try to find an answer to that.

This work by (Mirzadeh et al., 2019) claims that knowledge distillation is not effective when gap (in size) between teacher and student is large. Their experiments showed that a student model distilled from a teacher with more parameters and better accuracy performs worse than the same distilled from a smaller teacher with a smaller capacity. Specifically, they experimented with a fixed student network that was distilled from progressively growing(in size) teacher networks. It was observed that student's performance initially improved as the teacher became more complex, reached a peak and then fell with further growth in teacher size.

To explain this phenomenon, authors intuitively described certain factors that competed against each other when the teacher network was gradually made more complex. Initially, teacher's performance improved as its architecture becomes more complex, and was therefore able to provide better supervision to the student by being a more accurate predictor, thereby improving student performance. After a while, the teacher became so complex that the student lacked sufficient capacity or mechanics to mimic the teacher's behavior despite receiving hints, and its performance saturated. As the teacher size grew further, its certainty about data increased, thus making the logits less soft. This effectively weakened knowledge transfer via soft targets.

Clearly, factor 1 is in favor of increasing distillation performance while factors 2 and 3 are against it. Initially, as the teacher size increases, factor 1 prevails; as it grows further, factors 2 and 3 dominates. As a solution, authors proposed the use of *Teacher Assistant* (TA) networks that were intermediate to student and teacher in terms of size. Following a 2-step distillation process, the TA was first

trained from teacher followed by student being trained from the TA. They made two important observations regarding this training paradigm. Firstly, the best TA for training a student, was the one that lay in the middle not in terms of size but in terms of average accuracy. They found, while training a 2-layered student network, that the 4-layered TA worked best for distillation as it lay at the centre in terms of accuracy and not the 6-layered network that lay exactly in the middle of the student and the 10-layered teacher, in terms of size. And, although it was possible to train the TA network from scratch and then use it for training the student, best distillation results were achieved when the TA was also trained via distillation from the teacher network.

Essentially, this work makes the all important point that the quality of student is not linearly related to the quality of teacher. Thus, the best teacher is not the one that has the most complex architecture or achieves the best prediction accuracy but one that provides most informative soft targets. This, *in-principle*, opens up the possibility of distilling a student without using a deep teacher network. We shall discuss more on this topic in a later section.

In the next section, we digress a little from our discussion so far to describe another form of regularization that has a slightly different formulation but relates quite strongly to knowledge distillation.

5. Label Smoothing Regularization

The central theme of our discussion so far has been the drawback of cross-entropy objective using one-hot targets in training. It has been discussed how models can get over-confident due to assigning too much importance to the true class label. Moreover, it widens the gap between true-class logit and all other logits. This, combined with the bounded gradients of cross-entropy objective, restricts adaptability. (Szegedy et al., 2016) proposed a way to improve the generalization capacity of deep networks by modifying the target distribution $q(i|x)$ to take the form

$$q'(i|x) = (1-\epsilon)\delta_{i,y} + \epsilon u(i)$$

where i is the logit index (x, y) is the data tuple. The most important component of this formulation is the function u that redistributes some probability mass to the false classes, thereby reducing the gap with respect to true class. A commonly followed technique is to deduct ϵ mass from true class and have it uniformly distributed across all false classes. While experimenting with 1000 classes of ImageNet dataset, authors achieved a consistent(across architectures) gain of 0.2% by setting $u(k) = 1/1000$ and $\epsilon = 0.1$.

The objective function for LSR is defined in terms of cross entropy as

$$\begin{aligned} L(q', p) &= -\Sigma_{i=1}^{N} \log p(i) q'(i) \\ &= (1-\epsilon)L(q, p) + \epsilon L(u, p) \end{aligned}$$

The second term penalizes the deviation of predicted label distribution from the given distribution u. This deviation can also be captured by ***KL***-divergence as $L(u, p) = D_{KL}(u||p) + L(u)$, where $L(u)$ is constant. So in essence, Label Smoothing Regularization (LSR) is a composite objective which has a weighted cross-entropy component that extracts information from labelled data and a KL divergence component to ensure that the predictions match a given distribution. Assuming complete freedom to choose any distribution, this formulation allows for incorporating domain knowledge (bias) in the learning process, allowing for stronger regularization. Also, a careful glance at the objective reveals that knowledge distillation is a special case of LSR in which, soft labels from a teacher network is used as u.

Label smoothing has since been a widely used "trick" to improve network performance, however, there has not been a thorough investigation about why and when label smoothing should work. (Müller et al., 2019) tries to shed light upon behavior of neural networks trained with label smoothing in an attempt to formalize its relationship to knowledge distillation. Authors make several important observations when applying label smoothing to knowledge distillation. They observe that in spite of achieving stronger generalization capacity, label smoothing impairs distillation, i.e., when teacher models are trained with label smoothing, student models perform worse. Their experiments revealed that label smoothing actually encourage the representations of training examples from the same class, learned by the penultimate layer of the network, to group in tight clusters. Due to the formation of these class-wise clusters, every example of one class has similar distances to examples from all other classes. This leads to loss of information in the logits about resemblances between instances of different classes, which is necessary for distillation, but does not hurt generalization. They further observe that mutual information between inputs and logits undergo a rapid increase in the beginning of training but then slowly decreased in networks trained with label smoothing, indicating that much of the information that could be used to discriminate between examples was lost as the clusters are formed.

This work therefore suggests that, although label smoothing acts as an effective regularizer, it is not suitable for use in scenarios where the objective is to train a teacher network for distillation. In the next section, we discuss an alternative approach which involves formulating LSR as generalized knowledge distillation and thereby doing away with the need for an explicit teacher.

6. Teacher-free Knowledge Distillation

This section combines the key concepts that we discussed so far, forming the crux of our critical review. Here, we describe certain experiments that investigate the possibility of distillation without a teacher. (Yuan et al., 2019) claim that knowledge distillation is more of a regularization technique than a mechanism for transfer of similarity information, popularly known as dark knowledge.

Their first set of experiments were performed to understand if distillation is possible only from a fully trained and more complex teacher. Firstly, De-KD or Defective KD that uses a poorly trained teacher to train a student. Second, Re-KD or Reversed KD that uses a shallower student model to train a deep teacher model. By definition of KD, both these teachers are supposed to be *bad* and therefore should not transfer the required dark knowledge, reducing student accuracy. On the contrary, it was found that in both cases the average performance of the student improves. Authors analyze knowledge distillation process under the LSR framework to explain these counter-intuitive results.

As discussed in the previous section, label smoothing objective takes the form

$$L_{LS} = (1-\alpha)CE(q,p) + \alpha KL(u,p)$$

and knowledge distillation objective is defined as

$$L_{KD} = (1-\alpha)CE(q,p) + \alpha KL(p^t,p)$$

where p^t is the soft distribution of teacher predictions. Comparing the two equations, it can be established easily that knowledge distillation is a learned LSR whereas label smoothing is an ad-hoc knowledge distillation, which can be described as a teacher model with random accuracy and temperature $T = 1$.

At higher temperatures, the distribution of teacher's soft targets in knowledge distillation is more similar to the uniform distribution used in label smoothing. Therefore, authors explained the experimental results of Re-KD and De-KD by hypothesizing that the soft targets for these models at high temperatures were very close to having an uniform distribution and thus had a regularizing effect on their respective students, thus improving their performance. That is why a student can enhance the teacher and a poorly-trained teacher can still improve the student model.

Now we come to the all important question that forms the motivation for this review. *Can we achieve distillation without a teacher?*. In their second set of experiments, the authors enforce two forms of teacher-free training. Self-KD is implemented as a 2-step process. First a network is trained directly on data to generate a model S^t. Setting this model as teacher, the network is trained again from scratch, this time using distillation objective. Alternatively, a target distribution can be hand-crafted based on the specific task at hand. This is called Virtual-KD. For high values of T, this method converges to LSR. It is interesting to note that, both these mechanisms do as good as and sometimes better than teacher-based knowledge distillation. Results are shown in Table 1 & 2.

Model	Self-KD	Normal-KD [Teacher]
MobileNetV2	+2.58	+2.67 [ResNet18]
ShuffleNetV2	+1.89	+1.71 [ResNet18]
ResNet18	+1.23	+1.19 [ResNet50]
GoogLeNet	+1.45	+1.39 [ResNeXt29]
DenseNet121	+1.22	+1.15 [ResNeXt29]
ResNeXt29	+1.05	+1.12 [ResNeXt101]

Table 1. Comparison of accuracy improvement due to Self-KD (in %) on CIFAR100

For virtual-KD the target distribution was formulated as

$$p(i) = \begin{cases} a & \text{if } i = c, \\ \frac{1-a}{N-1} & \text{if } i \neq c, \end{cases}$$

where c is the true class index. $a = 0.99$ for the given experiments.

Model	Virtual-KD	Normal-KD [Teacher]
MobileNetV2	+2.50	+2.67 [ResNet18]
ShuffleNetV2	+1.75	+1.71 [ResNet18]
ResNet18	+1.49	+1.32 [ResNet50]
GoogLeNet	+1.07	+0.99 [ResNeXt29]

Table 2. Comparison of accuracy improvement due to Virtual-KD (in %) on CIFAR100

In addition to the results presented by the authors, we performed a set of experiments to empirically validate the effectiveness of teacher-free distillation. Our experimental setup and results are presented in the next section.

7. Experiments

For our experiments we design a test-bed as described in (Yuan et al., 2019). We use ShuffleNetV2 (Ma et al., 2018) as student and ResNet18 (He et al., 2016) as teacher. All the experiments were performed on CIFAR-10 dataset. We used Adam optimizer with a learning rate of $1e-3$. Our experiments were directed towards closing the gap between student accuracy achieved with the help of teacher and without by using multiple variants of label smoothing.

Method	Accuracy
Baseline	91.74
Normal-KD [RestNet18]	**92.86**
Uniform distribution with peak = 0.90	92.21
Geometric distribution with peak = 0.90	92.30
Uniform distribution with peak = 0.99	92.51
Reduced gap between highest and second highest classes	92.55
Geometric distribution with peak = 0.99	**92.71**

Table 3. Comparison of accuracy for multiple methods

7.1. Results

7.2. Discussion

In all our experiments, we used a composite loss weighted by a hyper-parameter α. Keeping the cross entropy loss consistent across all experiments, we present the variants of soft loss in Table 3. As shown, the student model trained with knowledge distillation performs better than the one trained on data. By removing the teacher, we applied several soft targets to regularize the network. In our experiments, geometric distribution worked best when peaked at true class. Intuitively, the geometric distribution offers smoother decay in probability mass from true class to other classes, as compared to a sudden jump in uniform distribution, which helps the network to capture the similarity information better. We further observed that, there was a gain in performance as the gap between the highest class and the next most confident class was reduced. This not only made the network less confident of the true class but also helped it to learn association between similar classes. Although, our results did not exactly match the performance achieved by the network under knowledge distillation, they clearly showed that it is possible to achieve comparable accuracy without an explicit teacher.

Therefore, it can be empirically shown that label smoothing regularization with a carefully designed objective function, can be a viable alternative to a cumbersome deep network based, multi-step optimization.

8. Adaptive Regularization of Labels

In this section, we present an alternative interpretation of the composite objective that has been central to our discussion so far. (Ding et al., 2019) found the optimization goals of the soft KL-divergence loss and the hard cross-entropy loss as contradictory. Considering z as logits and p as output predictions of student, q as one-hot labels, $q^{(soft)}$ as softened teacher predictions with temperature T and weight α, the gradient of the composite loss takes the form

$$\frac{dL_{KD}}{dz_i} = (1-\alpha)\frac{dL_{hard}}{dz_i} + \alpha\frac{dL_{soft}}{dz_i}$$

where $\frac{dL_{hard}}{dz_i} = (p_i - q_i)$ and $\frac{dL_{soft}}{dz_i} = \frac{1}{T}(p_i - q_i^{(soft)})$.

To make the objective 0, $\frac{dL_{KD}}{dz_i} = 0$, p_i takes the form,

$$p_i = \frac{(1-\alpha)q_i + \alpha T q_i^{(soft)}}{1-\alpha+\alpha T}$$

From this expression, it is clear that an optimal trade-off is required in terms of α and T for matching the soft probabilities, which may be hard to find. Further, as their values are fixed during training, the adaptability of the student model is restricted when soft labels are not available. To address these issues, they proposed an adaptive regularization scheme that enabled a neural network to self-learn correlation among classes using erroneous knowledge from the previous training experience.

To this end, they defined a *residual label* $q^{(res)}$ for each class. A residual label is a softmax-normalized vector of dimension $N-1$, where N being number of classes. This vector is extracted from rows of a correlation matrix of classes, where each row represents a class and each column records the probability of an instance of that class being misclassified as an instance of another class. Therefore, residual label is an erroneous prediction probability distribution of a class. Authors hypothesized that by maintaining an explicit matrix of residual correlation among classes, and feeding this information back to the network at each training step, it can be forced to learn the correlation among classes, which would have a regularizing effect on the performance. Also, based on the model's predictions, the correlation matrix is be updated at each step. The objective functions are defined as

$$L_{res} = -\frac{1}{N-1}\Sigma_{i=1}^{N-1} q_i^{(res)} \log p_i^{(res)}$$
$$L_{upd} = -\frac{1}{N-1}\Sigma_{i=1}^{N-1} p_i^{(res)} \log q_i^{(res)}$$

where $p^{(res)}$ is the erroneous probability distribution for a mini-batch. Here, L_{upd} is the update loss that is used to update the residual correlation matrix and the residual loss $L^{(res)}$ in addition to a hard CE loss is used to train the network. Compared to soft loss, residual loss doesn't have any hyper-parameter that require manual adjustment, thus automating the transfer of similarity information. Authors report superior results on multiple supervised learning tasks. For example, on CIFAR-100 using ResNet18 architecture with adaptive regularization, an improvement of 1.54% in accuracy was observed. Therefore, this work makes the

point that by utilizing the rich information content of erroneous predictions, a significant improvement in performance can be achieved without using an explicit teacher model.

9. Conclusion

In this work, we tried to understand the intuition and theory behind knowledge distillation and presented some cases where it requires improvement. We then discussed the similarities of knowledge distillation with label smoothing regularization and analyzed the idea that knowledge distillation is mainly a regularization technique and certain appropriately designed training objectives preclude the need for training a deep cumbersome network for distillation. We believe there is enough scope for theoretical investigations in this direction. It would be particularly useful to have an information theoretic analysis of networks that learn directly from data and others which learn from soft targets, providing interesting insights into the fundamental learning process. Another useful direction would be to study the robustness of these techniques under adversarial attacks.

References

Ba, L. J. and Caruana, R. Do deep nets really need to be deep? *CoRR*, abs/1312.6184, 2013. URL http://arxiv.org/abs/1312.6184.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In *International conference on machine learning*, pp. 2990–2999, 2016.

Ding, Q., Wu, S., Sun, H., Guo, J., and Xia, S.-T. Adaptive regularization of labels. 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the Knowledge in a Neural Network. *arXiv e-prints*, art. arXiv:1503.02531, Mar 2015.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 116–131, 2018.

Mirzadeh, S.-I., Farajtabar, M., Li, A., and Ghasemzadeh, H. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *arXiv preprint arXiv:1902.03393*, 2019.

Müller, R., Kornblith, S., and Hinton, G. When does label smoothing help? *arXiv preprint arXiv:1906.02629*, 2019.

Nakkiran, P., Alvarez, R., Prabhavalkar, R., and Parada, C. Compressing deep neural networks using a rank-constrained topology. 2015.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Yuan, L., Tay, F. E. H., Li, G., Wang, T., and Feng, J. Revisit Knowledge Distillation: a Teacher-free Framework. *arXiv e-prints*, art. arXiv:1909.11723, Sep 2019.

Adversarial Machine Learning and Certified Defenses against Adversarial Examples

Maomao Ding
Department of Statistics
Rice University
Houston, TX 77005
md45@rice.edu

Wei Wu
Department of Computational and Applied Mathematics
Rice University
Houston, TX 77005
weiwu995@gmail.com

Abstract

In recent years there has been an increasing interest in the topics of adversarial robustness in the deep learning community. Despite the outstanding accuracy on tasks such as visual recognition and speech recognition, deep learning classifiers are vulnerable to adversarial attacks. A small perturbation to the input data, often noise-like and imperceptible to human's eyes, could lead to an entirely different classification result. Researchers have proposed defense methods, which however are usually overcome by more advanced attack methods later. It is hence desirable to have defenses that are successful to all attacks within a certain class. However it is computationally intractable to even compute the worst-case error for any give network against all adversarial perturbations. Two common approaches are to approximate by minimizing a lower-bound of the worst case loss and to compute the worst-case perturbation exactly with discrete optimizations. These approaches suffer either from a low accuracy due to the bound being loose, or from the intractability of exact computation. In this paper, we discuss the basic concepts and methodology in adversarial robustness, then we discuss two methods to train deep neural network based classifier that are certified to be robust against norm-bounded perturbations in further details.

1 Introduction

Adversarial machine learning is an emerging field of study that gains its popularity in recent years. Machine learning techniques are originally designed for stationary data distribution and friendly environments where the trained models are deployed to work with test data generated from the same distribution. However, in real world applications, malicious adversaries may take advantage of these stationarity and attacks the learning system in all possibles phases. For example, poisoning (causative) attack is a class of attack that happens on the training phase. The attackers can influence or corrupt the ML model by 1). insert certain instances, 2). modify instances in the data, or 3). selectively remove certain instances. This type of attack does not happen frequently, as getting access to the training data and altering the training data distribution is usually difficult and not cost effective. Another type of attack, evasion (exploratory) attack, is the prevalent type of attack that the ML systems may encounter in real world applications. It does not alter the ML model, but instead, modifies and disguises the malicious objects to evade detection from the ML model. For example, adding some specific words in the spam email to lower the probability of being blocked by mail filtering system. Other possible attacks includes model inversion attack and model extraction attack, where the attackers try to extract sensitive data or model parameters from the ML model [1, 2]. Details of attack taxonomy can be found in [3, 4, 5].

Preprint. Under review.

In this exposition, we will mainly focus on evasion attack, especially in the deep neural network model class. People used to believe that trained deep learning models will be robust to data perturbation, i.e., slightly distorting input data will not change the predicted outcome. However, as pointed out by [6], the input-output mappings learning by deep neural networks are quite discontinuous, and we can easily deceive the DNN classifier by adding some unnoticeable noise to the original input. As is pointed out the authors, these perturbations are not random artifacts of learning: the perturbations have transferability both cross model and cross training set, meaning that the same perturbed sample can fool a different network, that was trained on a different training set. This intriguing discovery attracted a lot of attentions from the deep learning community, and exploding literatures have been published to discuss about defense and attack strategies.

Write $h_\theta : \mathcal{X} \to \{1, ..., K\}$ as the model that map inputs to the predicted categories, where K is the number of classes being predicted, and θ represents all the parameters contained in the model. We further define $\ell : \mathbb{R}^k \times \{1, ..., K\} \to \mathbb{R}$ as the loss function. For classification problem, the loss function is usually defined as cross entropy loss,

$$\ell(h_\theta(x), y) = \log \left[\sum_{j=1}^{K} \exp\{h_\theta(x)_j\} \right] - h_\theta(x)_y,$$

for $x \in \mathcal{X}$ and $y \in \{1, ..., K\}$ as the true class, and $h_\theta(x)_j$ denotes the j-th elements of the vector $h_\theta(x)$. For a certain data point (x, y), the adversarial example is usually found by maximizing $\ell(h_\theta(x + \delta), y)$ such that δ is an allowable perturbation. It is, however, difficult to characterize the set Δ of allowable perturbations. Ideally speaking, we want to capture any distortions that is imperceptible visually to humans. This could include anything from image noise, including Gaussian noise, salt-and-pepper noise, shot noise, to any form of image deformations, like rotation, scaling, barrel distortion, pincushion distortion and so on. It is impossible to give a mathematically rigorous definition that incorporates all the plausible perturbations, instead, we consider only some subset of the allowable set, such that a adversarial example could be found, or the trained model is robust against this perturbation. A commonly used perturbation set, is the norm-bounded adversarial perturbations, defined by

$$\Delta = \{\delta : \|\delta\| \leq \epsilon\},$$

where $\|\cdot\|$ can be any norm. Some frequently used norms are ℓ_∞, ℓ_1 and ℓ_2 norm.

Now suppose the data is generated from the distribution $\mathbb{P}$, the expected risk of h_θ is given as

$$R(h_\theta) = \mathbb{E}_{(x,y)\sim\mathbb{P}} \left[\ell(h_\theta(x), y) \right].$$

This quantity is usually unobservable. When a data set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ sampled from $\mathbb{P}$ is given, the traditional process usually wants to minimize the empirical risk

$$\widehat{R}(h_\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_\theta(x_i), y_i),$$

which is known as empirical risk minimization (ERM) principle in statistical learning theory.

When the model is subject to adversarial attack, we consider the adversarial risk as an alternative. This is defined as the expected value of maximum risk under the circumstance that the data is subject to perturbation. Formally speaking, we have

$$R_{adv}(h_\theta) = \mathbb{E}_{(x,y)\sim\mathbb{P}} \left[\max_{\delta\in\Delta(x)} \ell(h_\theta(x + \delta), y) \right],$$

where we allow the perturbation set Δ to be data-dependent. There is some concern about the measurability of the function $\max_{\delta\in\Delta(x)} \ell(h_\theta(x + \delta), y)$. Generally speaking, when $\Delta(x)$ is well-behaved, this function is usually measurable. When it is not measurable, concepts like outer expectation and envelops should be used. However, this complication does not give us too much insight about the problem, and from now on we will assume that $\max_{\delta\in\Delta(x)} \ell(h_\theta(x + \delta), y)$ is measurable without further verification.

Similar to the empirical risk, we can define the empirical adversarial risk as

$$\widehat{R}_{adv}(h_\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\delta_i\in\Delta(x_i)} \ell(h_\theta(x_i + \delta_i), y_i).$$

From attacker's perspective, building adversarial example from existing data point (x, y) is equivalent to find

$$\delta = \text{argmax}_{\delta \in \Delta(x)} \ell(h_\theta(x+\delta), y).$$

Sometimes the attacker would like create adversarial examples that the model would classify as y_{target}, not some random label among $\{1, ..., K\}$. This is known as targeted attack, and it is equivalent to solve the optimization problem

$$\begin{aligned} &\text{maximize}_{\delta \in \Delta(x)} \left\{ \ell(h_\theta(x+\delta), y) - \ell(h_\theta(x+\delta), y_{target}) \right\} \\ &\equiv \text{maximize}_{\delta \in \Delta(x)} \left\{ h_\theta(x+\delta)_{y_{target}} - h_\theta(x+\delta)_y \right\}. \end{aligned}$$

Note that typically we do not need to accurately solve this optimization problem. A feasible point that gives rise to significant raise in loss value should suffice. Based on the amount of knowledge the attacker has about the model structure, this could be further classified into: 1). white-box attack: the attacker has almost full knowledge about the algorithm, including model architecture and hyperparameters, etc. 2). black-box attack: the attacker almost know nothing about the algorithm, he gains his knowledge about the model mostly by feed data into the trained model that could give predicted outcomes. People used to believe that black-box attack should be difficult. However, as pointed out in [7], there exists extremely effective way of conducting black box attacks. As to the white-box attack, there exists many efficient ways of finding adversarial examples, including fast gradient sign method [8], DeepFool [9] and CW [10] among others.

From defender's perspective, it is crucial to train a classifier to the possible attacks. Formally speaking, the defender is trying to solve the following min-max optimization problem

$$\text{minimize } \widehat{R}_{adv}(h_\theta) \equiv \text{minimize } \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y) \in \mathcal{D}_{train}} \max_{\delta \in \Delta(x)} \ell(h_\theta(x+\delta), y),$$

which is also referred to as robust optimization formulation of adversarial learning.

Suppose that we are able to find f and g, such that $f(\theta) \leq \widehat{R}_{adv}(h_\theta) \leq g(\theta)$ for each θ. The commonly used defense techniques could be categorized into: 1). minimize $f(\theta)$, which minimizes the lower bound of empirical adversarial risk [11, 8]. This provides some protect again certain types of attack. However, this type of method could not guarantee that the model will be free of adversarial attacks, and countermeasures were quickly developed [10, 12]. 2). solve the min-max problem exactly [13, 16]. This could be done by formulating the inner maximization problem as a mixed integer linear programming problem. However, the computational burden is heavy for this scheme, and generally infeasible for complex networks used in practice. 3). minimize $g(\theta)$ [14, 15]. This strategy gives us certification that the trained network will be robust against all allowable perturbations, and the error is guaranteed to be within a certain value. This method is conservative, and the certified error bound is usually too large to be satisfying in practice.

2 Solving the inner maximization

In this section, we discuss the problem of solving the inner maximization, i.e.,

$$\text{maximize}_{\|\delta\| \leq \epsilon} \ell(h_\theta(x), y).$$

This problem is essential in adversarial learning. Adversarial examples are built by searching for optimal (suboptimal) points, and solving the inner maximization problem is usually the first step of building robust deep neural networks.

For simplicity, we consider a d-layer feed-forward network here.

$$\begin{aligned} z_1 &= x, \\ z_{i+1} &= f_i(W_i z_i + b_i),\ i = 1, ..., d \\ h_\theta(x) &= z_{d+1}, \end{aligned}$$

where f_i denotes the activation function at layer-i, which is usually taken as ReLU operator for $i = 1, ..., d-1$ and identify operator $f_i(z) = z$ for layer d. The parameters for this networks are $\theta = \{W_1, b_1, ..., W_d, b_d\}$.

2.1 Lower bounding the inner maximization

To find a lower bound of the inner maximization, or to propose a good adversarial example, gradient based methods are usually the most natural choice. Write the gradient of $\ell(h_\theta(x+\delta), y)$ with respect to δ as

$$v := \nabla_\delta \ell(h_\theta(x), y).$$

To maximize the loss, we would like to adjust x in the direction of v, i.e.,

$$x_{adv} := x + \alpha v,$$

where α is the step size. As a starting point, we would consider only conduct a one-step update, and in order to maximize the loss increments, we want the step size to be as large as possible. Since we have additional constrained that $\|\delta\| \leq \epsilon$, we also need to project αv back into this norm ball. When ℓ_∞ norm is used, this strategy gives us the Fast Gradient Sign Method (FGSM) [8],

$$\delta := \epsilon \cdot sign(v).$$

A natural extension would be using several gradient steps to perform a finer search in the allowable set. Since δ is restricted to a norm ball of radius ϵ, we will actually perform projected gradient descent (PGD). Notice that we are actually solving a maximization problem here, but by convention we will still call it gradient descent. Starting with $\delta = 0$, the PGD iterates are given as

$$Repeat:$$
$$\delta := \mathcal{P}(\delta + \alpha \nabla_\delta \ell(h_\theta(x+\delta), y)),$$

where $\mathcal{P}$ denotes the projection to the norm ball. DeepFool [9] used a similar strategy, except the slight difference that their goal is to find adversarial examples with minimum distance with respect to ℓ_2 norm. Along this line, gradient based methods like steepest descent or Nesterov's acceleration could also be used.

2.2 Solving the inner maximization exactly

There are many works using exact solvers to explore the properties of the neural networks. Two types of techniques are usually deployed. The first line of research uses Satisfiability Modulo Theories (SMT) [17, 18], and the second line of research formulate adversarial learning as integer programming problems [13, 16]. In this exposition, we will mainly focus on integer programming formulation of adversarial robustness.

Suppose that $W_i z_i + b_i$ can be bounded by l_i and u_i element-wisely. It is easy to reformulat $z_{i+1} = max\{0, W_i z_i + b_i\}$ as

$$\begin{aligned} z_{i+1} &\geq W_i z_i + b_i \\ z_{i+1} &\geq 0 \\ u_i \cdot v_i &\geq z_{i+1} \\ W_i z_i + b_i &\geq z_{i+1} + (1 - v_i) l_i \\ v_i &\in \{0,1\}^{|v_i|}, \end{aligned}$$

where v_i is a binary variable of the same size as z_{i+1}. Plug in this equivalent formulation into the original inner maximization problem, we obtain a mixed integer linear programming (MILP) problem, which can be solved using Gurobi or other solvers. The difficulty lies in the fact that each v_i can take $2^{|z_{i+1}|}$ many values, which imposes the combinatorial nature of this problem.

Notice the upper and lower bound will always exist, as we can bound each layer by extremely large intervals, e.g., $[10^{-100}, 10^{100}]$. However, the efficiency of the solver will depend heavily on these bounds. Notice that the input has some natural bound. We could propagate this bound layer by layer. Suppose $l \leq z \leq u$, then

$$\max\{W, 0\} l + \min\{w, 0\} u + b \leq Wz + b \leq \max\{W, 0\} u + \min\{w, 0\} l + b,$$

gives a bound for $Wz + b$. This bound is quite loose, as it element-wisely finds the bounds for $Wz + b$. when applying this trick to deep forward network, the bounds will getting worse as the layers stack up.

To certify robustness, we need to run the MILP using a targeted attack for all possible alternative classes for each data point. This further restrict the scalability of exact maximization approach.

2.3 Upper bounding the inner maximization

As the exact search for inner maximization problem for large scale networks will never terminate in reasonable time, it is of practical value to establish upper bound for inner maximization that is fast to obtain, and this will provide certificates on how our model behave under adversarial attacks.

A natural strategy is convex relaxation [19]. Notice that in the previous MILP formulation, the majority difficult is searching over $v_i \in \{0,1\}^{|v_i|}$, which grows exponentially with the number of units. One commonly used technique is to relax this constraint and allow v_i to take fractional values, i.e., $0 \leq v_i \leq 1$. With this modification, the inner maximization problem becomes a convex optimization problem, which can be solved by solvers like CVX. Notice that in this case, the obtained solution will not give us adversarial examples anymore, as the we've already altered the optimization problem. Here we are relaxing the optimization problem from the primal view. Actually we can also relax the optimization problem from the dual view. By weak duality, this also gives us an upper bound on the inner maximization problem, and under some mild conditions, it can be strong duality holds, which further shrink this upper bound with the actual optimal value. Further detail can be found in [19].

[20] used another approach to obtain an upper bound. Their approach is specifically designed for two layer feed forward networks. For simplicity we will consider a binary classification problem. The results obtained for binary classification can be easily adapted to multi-class classification case.

Write $h_\theta(x) = V\sigma(Wx)$, where $W \in \mathbb{R}^{m\times d}$ and $V \in \mathbb{R}^{2\times m}$ are the parameters of the first and second layer (including intercept), and σ is the activation function. For input x, we assume the true label for x is 2 and write $f(x) = h_\theta(x)_1 - h_\theta(x)_2 = V_1\sigma(Wx) - V_2\sigma(Wx)$, where V_i is the i-th row of V. Write $B_\epsilon(x)$ as the norm ball of radius ϵ centered at x. The maximizer for the inner maximization problem is $x_{adv} = \text{argmax}_{\widetilde{x}\in B_\epsilon(x)} f(x)$. When $f(x_{adv}) > 0$, we successfully find a adversarial example. Based on first order approximation, we have

$$f(\widetilde{x}) \approx f(x) + \nabla f(x)^T(\widetilde{x} - x) \leq f(x) + \epsilon\|\nabla f(x)\|_*,$$

where $\|\cdot\|_*$ is the dual norm. However, this approximated upper bound fails to hold in many cases, and some attacks can produce adversarial examples if the defense based on above approximation is used [10, 12]. Thus we use the following formula to obtain exact upper bound instead,

$$\begin{aligned} f(\widetilde{x}) &= f(x) + \int_0^1 \nabla f(t\widetilde{x} + (1-t)x)^T(\widetilde{x} - x)dt \\ &f(x) + \max_{\widetilde{x}\in B_\epsilon(x)} \epsilon\|\nabla f(\widetilde{x})\|_*. \end{aligned}$$

The RHS for the above formula is still intractable. We now work on further relax this bound. Write $v = V_1 - V_2 \in \mathbb{R}^m$. We further assume that the gradient for activation function is bounded. This holds for commonly used options like ReLU, sigmoid and tanh. For ReLU, we have $\sigma'(z) \in [0,1]$. Here we will work with ℓ_∞ norm. Thus $\|z\|_* = \|z\|_1 = \max_{t\in[-1,1]^d} t^T z$. Therefore:

$$\begin{aligned} \|\nabla f(\widetilde{x})\|_1 &= \|W^T\text{diag}(v)\sigma'(W\widetilde{x})\|_1 \\ &\leq \max_{s\in[0,1]^m} \|W^T\text{diag}(v)s\|_1 \\ &= \max_{s\in[0,1]^m, t\in[-1,1]^d} t^T W^T\text{diag}(v)s. \end{aligned}$$

The above bound is still non-convex. Similar to the MAXCUT problem, we approximate this by semidefinite programming relaxation.

First, we symmetrize the variable s, which gives

$$\max_{s\in[-1,1]^m, t\in[-1,1]^d} \frac{1}{2} t^T W^T\text{diag}(v)(\mathbf{1} + s).$$

Next we write the above quantity in quadratic form:

$$y := \begin{bmatrix} 1 \\ t \\ s \end{bmatrix} \quad M(v,W) := \begin{bmatrix} 0 & 0 & \mathbf{1}^T W^T\text{diag}(v) \\ 0 & 0 & W^T\text{diag}(v) \\ \text{diag}(v)^T W\mathbf{1} & \text{diag}(v)^T W & 0. \end{bmatrix}$$

The objective can be written as

$$\max_{y\in[-1,1]^{m+d+1}} \frac{1}{4} y^T M(v,W) y = \max_{y\in[-1,1]^{m+d+1}} \frac{1}{4} \langle M(v,W), yy^T \rangle.$$

Defining $P = yy^T$, we have $P \succeq 0$ and $P_{ii} \leq 1$. We obtain the following convex semidefinite relaxation of the original problem:

$$\max_{P\succeq 0, \mathbf{diag}(P)\leq 1} \frac{1}{4} \langle M(v,W), yy^T \rangle.$$

For multi-class problem, we write $f^{ij}(x) = h_\theta(x)_i - h_\theta(x)_j$. The rest is readily established.

The last approach we present here, is the interval bound propagation (IBP) [20], which is simple to implement and scales up well to large scale networks.

The idea is similar to what we do to establish upper and lower bounds in building up MILP for exact solution of inner maximization. In fact, there are immediately available upper bounds for any linear combination of $h_\theta(x)$, which we write as $c^T h_\theta(x)$. However, we know this bound is quite loose. To further improve this bound, we can estimate the lower and upper bound for up to the second last layer, and minimize $c^T h_\theta(x)$. Suppose that $l \leq z_d \leq u$, we have

$$\begin{aligned} &\text{minimize}_{z_d} c^T(W_d z_d + b_d) + c^T b_d = (W_d^T c)^T z_d + c^T b_d \\ &\quad \text{subject to} l \leq z \leq u. \end{aligned}$$

There is a closed form for the optimal value

$$\max\{c^T W_d, 0\} l + \min\{c^T W_d, 0\} u + c^T b_d.$$

This bound can be efficiently computed. We can easily use this to maximize loss or targeted loss for a certain data point.

3 Solving the outer minimization

In this section, we discuss the problem of solving the outer minimization problem, i.e., training networks that can defense adversarial attacks. When the inner maximization can be solved exactly, the gradient of $\max_{\delta\in\Delta(x)} \ell(h_\theta(x), y)$ can be given by Danskin's theorem. Write

$$\delta^*(x) = \text{argmax}_{\delta\in\Delta(x)} \ell(h_\theta(x), y)$$

which is the maximizer of the inner maximization problem. The gradient of the adversarial loss is

$$\nabla_\theta \max_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta), y) = \nabla_\theta \ell(h_\theta(x+\delta^*(x)), y).$$

This facilitates the mini-batch gradient descent for empirical adversarial loss

$$\theta := \theta - \frac{\alpha}{|\mathcal{B}|} \sum_{(x,y)\in\mathcal{B}} \nabla_\theta \ell(h_\theta(x+\delta^*(x)), y),$$

where α is the step size and $\mathcal{B}$ is the mini-batch. However, finding exact solution of the inner maximization is extremely time-consuming, rendering the outer minimization impractical. Thus we will focus on minimizing the lower and upper bound of inner maximization problem.

3.1 Minimizing the lower bound

Write $x + \widetilde{\delta}(x)$ as the adversarial example created by algorithm, e.g., FGSM. The gradient step can be written as

$$\theta := \theta - \frac{\alpha}{|\mathcal{B}|} \sum_{(x,y)\in\mathcal{B}} \nabla_\theta \ell(h_\theta(x+\widetilde{\delta}(x)), y).$$

Strictly speaking, Danskin's theorem only applies at $\delta = \delta^*(x)$, and it's not clear about what the properties we would have when computing the gradient at suboptimal points. In practice, we find that the "quality" of such a gradient descent scheme is closely related to the quality of inner maximization step, and the trained model will be robust against to the attack we used to train the inner maximization. However, when other form of attacks is presented, the trained model is likely to fail.

3.2 Minimizing the upper bound

Danskin's theorem is directly applicable here. In the case of convex relaxation or IBP, the gradient is easy to compute. One thing we need to be cautious about is that we should not directly optimize with our targeted ϵ. In many cases, this causes our model to collapse, predicting equal probability for every class. Instead, we should start with small ϵ, then gradually increases to our aimed perturbation level.

For the semidefinite convex relaxation [15], finding the optimal value of a SDP is usually slow. We seek to the duality theory to alleviate the computation burden.

The primal problem is

$$\begin{aligned} &\text{maximize } \langle M, P\rangle \\ &\text{subject to } P \succeq 0, \text{diag}(P) \leq 1, \text{tr}(P) \leq d+m+1. \end{aligned}$$

Forming the Lagrangian for the constraints $\text{diag}(P) \leq 1$, we have the following equivalent problem

$$\begin{aligned} &\text{maximize } \min_{c\geq 0} \langle M, P\rangle + c^T(\mathbf{1} - \text{diag}(P)) \\ &\text{subject to } P \succeq 0, \text{tr}(P) \leq d+m+1. \end{aligned}$$

By strong duality of linear programming problem, this is equivalent to

$$\begin{aligned} &\text{minimize } \max_{P\succeq 0, \mathbf{tr}(P)\leq d+m+1} \langle M, P\rangle + c^T(\mathbf{1} - \text{diag}(P)) \\ &\text{subject to } c \geq 0. \end{aligned}$$

The inner maximum is equivalent to

$$\mathbf{1}^T c + (d+m+1)\left(\max_{P\succeq 0, \mathbf{tr}P\leq 1} \langle M - \text{diag}(c), P\rangle\right) = \mathbf{1}^T c + (d+m+1)\lambda_{max}(M - \text{diag}(c))_+,$$

where a_+ denote the positive part of a. Notice that $\lambda_{max}(M - \text{diag}(c)) = \max_{\|u\|_2=1} u^T(M - \text{diag}(c))u$ is an decreasing function with each components of c, and $\mathbf{1}^T cI(c \geq 0) = \mathbf{1}^T \max(c, 0)$. We can write the about problem as the following unconstrained optimization problem

$$\text{minimize } \mathbf{1}^T \max(c, 0) + (d+m+1)\lambda_{max}(M - \text{diag}(c))_+$$

Thus the objective function in its unconstrained (Lagrangian) form is:

$$\sum_{(x,y)\in\mathcal{D}} \ell(h_\theta(x), y) + \lambda \mathbf{1}^T \max(c, 0) + (d+m+1)\lambda_{max}(M(V, W) - \text{diag}(c))_+,$$

where λ is the tuning parameter control the trade-off between classification loss and adversarial robustness. This can be optimized efficiently with gradient based method, and $\lambda_{max}(\cdot)$ can be computed using efficient algorithms like Lanczos. The dual formulation also gives us a certificate on robustness. Specifically, suppose (W_t, V_t, c_t) are the obtained parameters after t-th iteration. The duality theory tells us

$$f(\widetilde{x}) \leq f(x) + \frac{\epsilon}{4}\left[\mathbf{1}^T \max(c_t, 0) + (d+m+1)\lambda_{max}(M(V_t, W_t) - \text{diag}(c_t))_+\right],$$

for any adversarial example $\widetilde{x}$.

References

[1] Fredrikson, Matt, Somesh Jha, and Thomas Ristenpart. "Model inversion attacks that exploit confidence information and basic countermeasures." In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1322-1333. ACM, 2015.

[2] Tramèr, Florian, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. "Stealing machine learning models via prediction apis." In 25th USENIX Security Symposium (USENIX Security 16), pp. 601-618. 2016.

[3] Huang, Ling, Anthony D. Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J. Doug Tygar. "Adversarial machine learning." In Proceedings of the 4th ACM workshop on Security and artificial intelligence, pp. 43-58. ACM, 2011.

[4] Biggio, Battista, Giorgio Fumera, and Fabio Roli. "Security evaluation of pattern classifiers under attack." IEEE transactions on knowledge and data engineering 26, no. 4 (2013): 984-996.

[5] Biggio, Battista, Igino Corona, Blaine Nelson, Benjamin IP Rubinstein, Davide Maiorca, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. "Security evaluation of support vector machines in adversarial environments." In Support Vector Machines Applications, pp. 105-153. Springer, Cham, 2014.

[6] Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks." arXiv preprint arXiv:1312.6199 (2013).

[7] Papernot, Nicolas, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. "Practical black-box attacks against machine learning." In Proceedings of the 2017 ACM on Asia conference on computer and communications security, pp. 506-519. ACM, 2017.

[8] Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." arXiv preprint arXiv:1412.6572 (2014).

[9] Moosavi-Dezfooli, Seyed-Mohsen, Alhussein Fawzi, and Pascal Frossard. "Deepfool: a simple and accurate method to fool deep neural networks." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2574-2582. 2016.

[10] Carlini, Nicholas, and David Wagner. "Towards evaluating the robustness of neural networks." In 2017 IEEE Symposium on Security and Privacy (SP), pp. 39-57. IEEE, 2017.

[11] Papernot, Nicolas, et al. "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks." 2016 IEEE Symposium on Security and Privacy, SP 2016. Institute of Electrical and Electronics Engineers Inc., 2016.

[12] Tramèr, Florian, et al. "Ensemble adversarial training: Attacks and defenses." arXiv preprint arXiv:1705.07204 (2017).

[13] Tjeng, Vincent, Kai Xiao, and Russ Tedrake. "Evaluating robustness of neural networks with mixed integer programming." arXiv preprint arXiv:1711.07356 (2017).

[14] Wong, Eric, and J. Zico Kolter. "Provable defenses against adversarial examples via the convex outer adversarial polytope." arXiv preprint arXiv:1711.00851 (2017).

[15] Raghunathan, Aditi, Jacob Steinhardt, and Percy Liang. "Certified defenses against adversarial examples." arXiv preprint arXiv:1801.09344 (2018).

[16] Cheng, Chih-Hong, Georg Nührenberg, and Harald Ruess. "Maximum resilience of artificial neural networks." International Symposium on Automated Technology for Verification and Analysis. Springer, Cham, 2017.

[17] Huang, Xiaowei, et al. "Safety verification of deep neural networks." International Conference on Computer Aided Verification. Springer, Cham, 2017.

[18] Katz, Guy, et al. "Reluplex: An efficient SMT solver for verifying deep neural networks." International Conference on Computer Aided Verification. Springer, Cham, 2017.

[19] Salman, Hadi, et al. "A convex relaxation barrier to tight robustness verification of neural networks." Advances in Neural Information Processing Systems. 2019.

[20] Gowal, Sven, et al. "On the effectiveness of interval bound propagation for training verifiably robust models." arXiv preprint arXiv:1810.12715 (2018).

Can Theoretical Algorithms Efficiently Escape Saddle Points in Deep Learning?

Sean Farrell [*1] **Carlos Quintero Peña** [*2]

Abstract

This document provides a literature review for the most important recent works related to optimizing high-dimensional non-convex functions in the presence of saddle points mostly for machine learning applications. The inspiration came from reviewing the paper "How to Escape Saddle Points Efficiently?" (Jin et al., 2017a). A large research effort has been devoted to proposed methods that can converge to second order stationary points efficiently. Of special interest is the set of methods that do not rely on Hessian computation, mainly driven by applications in machine learning where this may not be feasible. Although, many important theoretical results have been proposed, many of them have not be tested in real experiments, especially in the context of training a deep neural network. We have designed experiments with different network architectures and state-of-the-art datasets to observe the behavior of perturbed versions of gradient descent. Initial results show that an improvement in experimental convergence rate can be seen only for small and shallow networks. These results, although still encouraging, does not allows us to conclude on the practicality of the analyzed algorithms.

1. Literature Review

1.1. Context

The pioneering work of (Dauphin et al., 2014) is one of the first works that bring attention to the analysis of convergence of gradient-based optimization methods in the presence of high-dimensional non-convex functions in a variety of applications, including statistical physics and neural networks, among others. In general, the proliferation of saddle points as the dimension of the problem increases is exponential, suggesting that saddle points and not local minima are responsible for slowing down the convergence of both first order and second order methods. Their algorithm called saddle-free Newton method (SFN) uses curvature information to define a trust region allowing it to escape saddle points. Their observations are experimentally justified in the context of neural networks, however, no theoretical characterization is provided.

The intuition gained from experimentation in low-dimensional problems can lead us to misleading conclusions when extending the concepts to higher-dimensional problems. In an extreme case, consider the random matrix theory. Choosing an eigenvalue with exact eigenvalue of 0 has probability 0, while for larger dimensions it is exponentially unlikely to get all eigenvalues either positive or negative, which means that most critical points will be saddle points.

(Choromanska et al., 2014) shows one of the first theoretical attempts to explain the optimization of highly non-convex multi-layer neural network functions using results from the prism of spin-glass theory.

These results have inspired a plethora of works that aimed at: i) understanding the behavior of gradient descent (GD) under the presence of saddle points, ii) the proposal of GD modifications that aim at improving its convergence properties to converge to second-order stationary points and iii) the proposal of new algorithms that are not GD modifications that achieve improved performance over traditional optimization algorithms in the presence of saddle points. Most of these efforts have been heavily driven by applications in machine learning and signal processing, where non-convex and high-dimensional problems easily arise.

A highly influential line of work is the one that strives to understand and improve the behavior of GD-based algorithms in the presence of saddle points. Originally, (Ge et al., 2015b) showed how a simple variation of GD named Noisy Stochastic Gradient can converge to a local minimum in $d^4\text{poly}(\epsilon^{-1})$, where d is the dimension and ϵ is the size of the gradient at the critical point (see Definition 1). In their approach the idea is to add noise to the gradient sampled uniformly from the unit sphere. Their choice of adding noise

[*]Equal contribution [1]Department of Electrical and Computer Engineering, Rice university, Houston, Texas, USA [2]Department of Computer Science, Rice university, Houston, Texas, USA. Correspondence to: Sean Farrell <smf5@rice.edu>, Carlos Quintero Peña <carlosq@rice.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

to the gradient is based on observations that even though gradient-based methods will not move when converged to a stationary point (one with $\nabla f(x) = 0$), the randomness in stochastic gradient updates helps the algorithm to escape unstable stationary points, such as saddle points. In their proof, they use the fact that when the algorithm is near a saddle point, there is a finite number of steps in which the expected value of the function will slightly decrease with respect to the one in the stationary point. In other words, the update of the algorithm guarantees that the point will move in a direction of negative curvature and will remain close to it in directions of positive curvature. They prove this using martingale theory. Later, (Lee et al., 2016) showed that gradient descent converges asymptotically to minima with either random initialization or noise. For this, the authors use the Stable Manifold Theorem. Intuitively, if the initial point of the algorithm has a component outside the subspace spanned by the standard basis vectors corresponding to the positive eigenvalues of the Hessian, then GD will converge to the corresponding saddle point. The probability of the initial point landing in this subspace is zero. In their proof, they use the fact that there exists a diffeomorphism between the neighborhood of a critical point and a stable center manifold that contains the points that are locally forward not escaping. This results may apply even when the noise is not artificially added which means that variants of GD such as stochastic gradient descent also fall under this analysis. However, the noise coming from the stochastic gradients may not be good enough in the required directions.

(Levy, 2016) showed how a variant of normalized gradient descent (NGD) could be extended to efficient escape strict saddle points within $\mathcal{O}(\eta^{-2})$ which is an improvement to (Ge et al., 2015a) Noisy gradient descent (Noisy-GD) convergence guarantees. The proposed algorithm is called Saddle-NGD, and it adds zero mean Gaussian noise θn_t with a variance dependent on the dimensionality of the problem once every N_0 iterations. The main difference between this method and Noisy-GD is the fact that only the direction of the gradient is taken into account and the noise is sampled from a different distribution after a certain number of iterations. This method is beneficial because near a saddle point the gradients approach minute values near zero. The normalization step helps to ensure a fast escape from a saddle point because once the algorithm arrives at a saddle point the most negative eigenvalue will be sufficiently large relative to the other eigenvalues. Experimentally Levy (2016) tested his proposed Saddle-NGD algorithm against (Ge et al., 2015a) Noisy-GD method on online tensor data significant in big data applications. The results show that Saddle-NGD has slower initial convergence improvements, compared to Noisy-GD, but after a specific critical point dependent on the learning rate, Saddle-NGD shows significant improvements in decreasing the reconstruction error.

Work presented by (Du et al., 2017) showed that general gradient descent will require exponential time to escape strict saddle points in general non-convex smooth functions using natural random initialization methods (Du et al., 2017). They further tested (Jin et al., 2017a) PGD algorithm, showing that it outperforms GD by taking only polynomial time instead of exponential time to escape saddle points. Through theoretical and experimental work Du et al. (2017) showed that GD takes at least t_d exponential time to escape d saddle points following $t_d \geq (\frac{L+\gamma}{\gamma})^d$. Where L and γ are characteristics of the non-convex function being optimized. The PGD algorithm for the same experiments and theory required an approximately constant number of iterations of approximately $\frac{1}{\eta\gamma}$, where η is the specified learning rate. For general non-convex optimization problems (Du et al., 2017) showed the significant improvements PGD has versus GD with random initialization to reduce the convergence time requirements from exponential time to polynomial time. This can be significant for high dimensional neural network problems to help reduce training time. However, these results cannot be generalized to all non-convex problems, because there can be classes of problems and initialization schemes where GD can have better performance in the presence of saddle points (Du et al., 2017).

1.2. Perturbed Gradient Descent

Here is where the paper selected to review for this project comes in. In (2017a), Jin et al., proposed Perturbed Gradient Descent (PGD), a simple variation of GD capable of achieving convergence to ϵ-second order stationary points in a number of iterations that is almost "dimension-free", which means that its complexity depends only poly-logarithmically on the problem dimension. This result outperforms previous results for the following reasons:

- Previous works were able to characterize the convergence behavior of GD algorithms either asymptotically or bounding their complexity polynomially in terms of the problem dimensionality. This work attained improved results by providing sharp bounds for second order stationary points
- Their analysis leads to a convergence complexity that matches those of the original GD (Nesterov) up to a poly-logarithmic factor
- Their results apply to a more general class of non-convex functions instead of being problem-specific
- This work is an important step towards reducing the gap between practice and theory in this field, since it better characterizes the behavior observed in practice when optimizing high-dimensional non-convex functions using GD-based algorithms

The dynamics of the PGD algorithm are mostly that of the GD, except that it keeps track of the gradient's norm to identify when the current iteration is close to a stationary point. At that point, PGD adds noise to the current iterate by sampling uniformly from a d-dimensional ball, only a maximum amount of iterations. If the function value does not decrease enough, the algorithm is potentially in a local minimum and returns the current point. Conversely, if the function value decreases enough, it has escaped the saddle point and regular GD iterations are put in place again. The algorithm is shown in Algorithm 1.

Algorithm 1 Perturbed Gradient Descent (PGD)

Input: x_0, l, ρ, ϵ, c, δ, Δ_f
$\chi \leftarrow 3\max\{\log\left(\frac{dl\Delta_f}{c\epsilon^2\delta}\right), 4\}, \eta \leftarrow \frac{c}{l}, r \leftarrow \frac{\sqrt{c}}{\chi^2}\frac{\epsilon}{l}$,
$g_{thres} \leftarrow \frac{\sqrt{c}}{\chi^2}\epsilon . f_{thres} \leftarrow \frac{c}{\chi^3}\sqrt{\frac{\epsilon^3}{\rho}}, t_{thres} \leftarrow \frac{\chi}{c^2}\frac{l}{\sqrt{\rho\epsilon}}$
$t_{noise} \leftarrow -t_{thres} - 1$
for $t = 0, 1, ...,$ **do**
  if $\|\nabla f(x_t)\| \leq g_{thres}$ and $t - t_{noise} > t_{thres}$ **then**
    $\tilde{x} \leftarrow x_t, t_{noise}$
    $x_t \leftarrow \tilde{x}_t + \xi_t$, ξ_t uniformly $\mathbb{B}_0(r)$
  end if
  if $t - t_{noise} = t_{thres}$ and $f(x_t) - f(\tilde{x}_{t_{noise}}) > -f_{thres}$ **then**
    Return $\tilde{x}_{t_{noise}}$
  end if
  $x_{t+1} \leftarrow x_t - \eta\nabla f(x_t)$
end for

For further clarification, we briefly present the following definitions:

Definition 1. *For a differentiable function f, we say that x is a first-order stationary point if $\|\nabla f(x)\| = 0$; we also say x is an ϵ-first-order stationary point if $\|\nabla f(x)\| \leq \epsilon$.*

Definition 2. *For a differentiable function f, we say that x is a local minimum if x is a first-order stationary point, and there exists $\epsilon > 0$ so that for any y in the ϵ-neighborhood of x, we have $f(x) \leq f(y)$; we also say x is a saddle point if x is a first-order stationary point but not a local minimum. For a twice-differentiable function f, we further say a saddle point x is strict (or non-degenerate) if $\lambda_{min}(\nabla^2 f(x)) < 0$*

Definition 3. *For a ρ-Hessian Lipschitz function f, we say that x is a second-order stationary point if $\|\nabla f(x)\| = 0$ and $\lambda_{min}(\nabla^2 f(x)) \geq 0$; we also say x is ϵ-second-order stationary point if $\|\nabla f(x)\| \leq \epsilon$ and $\lambda_{min}(\nabla^2 f(x)) \geq -\sqrt{\rho\epsilon}$*

Note that these definitions define ϵ-first order and second order stationary points in terms of ϵ to make explicit the relation between the gradient and Hessian. Jin et al., (2017a) main results is stated below:

Theorem 1. (Jin et al., 2017a) *Assume that f satisfies that is l-smooth and ρ-Hessian Lipschitz. Then there exists an absolute constant c_{max} max such that, for any $\delta > 0$, $\epsilon \leq \frac{l^2}{\rho}, \Delta_f \geq f(x_0) - f^*$, and constant $c \leq c_{max}, PGD(x_0, l, \rho, \epsilon, c, \delta, \Delta_f)$ will output an ϵ-second-order stationary point, with probability $1 - \delta$, and terminate in the following number of iterations:*

$$O\left(\frac{l(f(x_0) - f^*)}{\epsilon^2}\log^4\left(\frac{dl\Delta_f}{\epsilon^2\delta}\right)\right). \tag{1}$$

In addition to this general result, the authors also show analysis for strict saddle property, strong convexity and provide examples of matrix factorization. The proof is based on a geometric interpretation of the perturbation ball achieved by the PGD algorithm. In general, the perturbation region is divided by two disjoint regions; the escaping region which contains all the points where the algorithm can escape the critical point and the stuck region where that is not possible. The shape of these regions is in general not known; however, the authors bound the volume of the stuck region using the smallest eigendirection of $\nabla^2 f(\tilde{x})$ as the thickness of such small band. In this way, they can guarantee that PGD will escape the critical points with high probability. Figure 1 shows graphically the described concept in $2D$ and $3D$. For the former, a thin band is created near the point where the perturbation is performed; in $3D$, it becomes a thin disk.

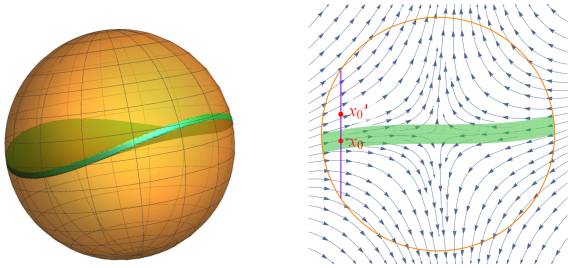


Figure 1. Ball region created by the perturbation in PGD algorithm showing the escaping and stuck region. The region in green corresponds to the stuck region and the authors show that its volume is small. Figure taken from (Jin et al., 2017a).

As stated above, this result shows only a poly-logarithmic convergence on the dimension, which can be written as $\tilde{O}\left(\epsilon^{-2}\right)$ and can be compared to the convergence of original GD (Nesterov) to achieve a ϵ-first order stationary point of $O\left(\frac{l(f(x_0)-f^*)}{\epsilon^2}\right)$, or equivalently $O\left(\epsilon^{-2}\right)$.

1.3. Perturbed Accelerated Gradient Descent

The natural way to improve on the PGD convergence rate involved momentum-based techniques. Previously, (Nesterov) showed that an accelerated version of gradient descent can achieve a $\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$ to an ϵ-first order stationary point which is faster than traditional gradient descent. Jin et al. (2017b) investigated the theoretical possibilities of developing a momentum-based algorithm that could achieve faster convergence then GD in the presence of strict saddle points. Their algorithm is called Perturbed Accelerated Gradient Descent (PAGD) with the pseudo-code presented in Algorithm 2 (Jin et al., 2017b). The first *if-statement* in PAGD is used for the perturbation step. A perturbation to the current position is added when the gradient is small and the previous perturbation was added over $\mathfrak{T}$ iterations ago. The noise is sampled uniformly from a d-dimension ball with radius r (Jin et al., 2017b). The following three lines correspond to the AGD component of the algorithm. The last *if-statement* in PAGD is used for the negative curvature exploitation (NCE). The NCE steps make sure the Hamilton will decrease per iteration. Intuitively this helps to restrict the momentum incorporated in PAGD from growing too large where negative curvature can not be exploited. Jin et al. (2017b), show that their PAGD method is one of the first Hessian-free single loop algorithms to find second-order stationary points faster than GD in $\mathcal{O}(1/\epsilon^{7/4})$ iterations (Jin et al., 2017b).

Algorithm 2 Perturbed Accelerated Gradient Descent (PAGD)

Input: initial point x_0, step sizeη, γ, s, r, $\mathfrak{T}$, and v_0.
for $t = 0, 1, ...,$ **do**
  if $\|\nabla f(x_t)\| \leq \epsilon$ and no perturbation in last $\mathfrak{T}$ steps **then**
    $x_t \leftarrow x_t + \chi_t \quad \chi_t = Unif(B_0(r))$
  end if
  $y_t \leftarrow x_t + (1-\theta)\,\mathrm{v}_t$
  $x_{t+1} \leftarrow y_t - \eta \nabla f(y_t)$
  $v_{t+1} \leftarrow x_{t+1} - x_t$
  if $f(x_t) \leq f(y_t) + \{\nabla f(y_t), x_t - y_t\} - \frac{\gamma}{2}\|x_t - y_t\|^2$ **then**
    if $\|v_t\| \geq s$ **then**
      $x_{t+1} \leftarrow x_t$
    else
      $\delta = s \cdot v_t / \|v_t\|$
      $x_{t+1} \leftarrow \mathrm{argmin}_{x \in (x_t + \delta, x_t - \delta) f(x)}$
      return $(x_{t+1}, 0)$
    end if
  end if
end for

1.4. Stochastic Gradient Descent Algorithm

The standard SGD algorithm does not require the complete gradient $\nabla f(\cdot)$ but instead uses a stochastic gradient $\mathbf{g}(\mathbf{x},\theta)$ when at some location $\mathbf{x}$. The θ term is a random variable sampled from a specific distribution $\mathcal{D}$. The SGD algorithm is proven to converge to an ϵ-first order stationary point in $\mathcal{O}(\epsilon^{-4})$ (Ghadimi & Lan, 2013). The general convergence for SGD is shown in Theorem 2.

Theorem 2. (Ghadimi & Lan, 2013) *Let the function f satisfy Lipschitz gradients and the stochastic gradient $\boldsymbol{g}$ has an expectation equal to the true gradient and the sampling distribution has strongly bounded tails. Let the step size scale as $\eta = \tilde{\Theta}(\ell^{-1}(1+\sigma^2/\epsilon^2)^{-1})$. Then with probability $1-\delta$, SGD will find an ϵ-first order stationary point in the following iteration shown in Eq.2.*

$$\tilde{\mathcal{O}}\left(\frac{\ell(f(x_0)) - f^*}{\epsilon^2} \cdot \left(1 + \frac{\sigma^2}{\epsilon^2}\right)\right) \tag{2}$$

The SGD algorithm pseudo-code is presented in Algorithm 3. This method can also be used in a mini-batch method where several random variables are sampled from the distribution and used to calculate and average stochastic gradient.

Algorithm 3 Stochastic Gradient Descent (SGD)

Input: initial point x_0, step size η.
for $t = 0, 1, ...,$ **do**
  sample $\theta_t \sim \mathcal{D}$
  $x_{t+1} \leftarrow x_t - \eta(\mathrm{g}(x_t;\theta_t))$
end for

1.5. Perturbed Stochastic Gradient Descent Algorithm

The PSGD algorithm is a variant of SGD that theoretically has been proven to converge to an ϵ-second-order stationary point in $\tilde{\mathcal{O}}(\epsilon^{-4})$ when the gradients are Lipschitz (Jin et al., 2019). If the Lipschitz assumption cannot be applied then a linear dimensional dependence d arises, bounding convergence to an ϵ-second-order stationary point in $\tilde{\mathcal{O}}(d\epsilon^{-4})$ (Jin et al., 2019). The complete theorem formulation of this convergence criterion is shown in Theorem 3.

Theorem 3. (Jin et al., 2019) *Let the function f satisfy Lipschitz gradients and the stochastic gradient $\boldsymbol{g}$ has an expectation equal to the true gradient and the sampling distribution has strongly bounded tails. For any ϵ, $\delta > 0$, the PSGD algorithm with chosen parameters (η, r) will find an ϵ-second order stationary point in the following number of iterations in Eq. 3, with probability $1-\delta$.*

$$\tilde{\mathcal{O}}\left(\frac{\ell(f(x_0)) - f^*}{\epsilon^2} \cdot \mathfrak{N}\right) \tag{3}$$

The parameters η, r, and $\mathfrak{N}$ are chosen based on Eq.4-5 by setting theoretical parameters about the optimized functional space.

$$\eta = \tilde{\Theta}\left(\frac{1}{\ell \cdot \mathfrak{N}}\right), \quad r = \tilde{\Theta}(\epsilon\sqrt{\mathfrak{N}}), \tag{4}$$

$$\text{where } \mathfrak{N} = 1 + \min\left\{\frac{\sigma^2}{\epsilon^2} + \frac{\tilde{\ell}^2}{\ell\sqrt{\rho\epsilon}}, \frac{\sigma^2 d}{\epsilon^2}\right\} \tag{5}$$

Recall that SGD finds convergence to an ϵ-first-order stationary point in $\mathcal{O}(\epsilon^{-4})$ (Ghadimi & Lan, 2013). Thus, PSGD can achieve the same convergence as SGD to a ϵ-second order stationary point with a poly-logarithmic dependence in dimension d. This is significant because convergence to second-order stationary points eliminates the possibility of the convergence point being a strict saddle point. However, that being said the convergence point could either be a local/global minimum or degenerate saddle point.

The general PSGD algorithm pseudo code developed by Jin eta al. (2019) is presented in Algorithm 4 and the mini-batch version is presented in Algorithm 5 (Jin et al., 2019). For both algorithms random noise is added to the gradient each iteration. The noise ξ_t is sampled from a normal distribution with zero mean and covariance $(r^2/d\mathbf{I})$. If r is selected as $r = \tilde{\Theta}(\epsilon)$ then, theoretically PSGD will find an ϵ-second order stationary point following Theorem 3.

Algorithm 4 Perturbed Stochastic Gradient Descent (PSGD)

Input: initial point x_0, step size η, perturbation radius r.
for $t = 0, 1, ...,$ **do**
  sample $\theta_t \sim \mathcal{D}$
  $x_{t+1} \leftarrow x_t - \eta(\mathrm{g}(x_t;\theta_t) + \xi_t), \quad \xi_t \sim \mathcal{N}(0, (r^2/d)\mathbf{I})$
end for

Algorithm 5 Mini-batch Perturbed Stochastic Gradient Descent (Mini-batch PSGD)

Input: initial point x_0, step size η, perturbation radius r.
for $t = 0, 1, ...,$ **do**
  sample $\{\theta_t^{(1)}, ..., \theta_t^{(m)}\} \sim \mathcal{D}$
  $g_t(x_t) \leftarrow \sum_{i=1}^{m} g(x_t; \theta_t^{(i)})/m$
  $x_{t+1} \leftarrow x_t - \eta(\mathrm{g}(x_t;\theta_t) + \xi_t), \quad \xi_t \sim \mathcal{N}(0, (r^2/d)\mathbf{I})$
end for

A very recent work by Fang et al., (2019) showed how SGD can benefit from added dispersive noise and converge to a ϵ-second order stationary point in $\tilde{\mathcal{O}}(\epsilon^{-3.5})$, which improves on the PSGD convergence rate and is the sharpest result so far for stochastic gradient-based methods. Its worth mentioning that these convergence results that are almost dimension free, holding when the gradients are Lipschitz continuous. When this condition does not hold, a linear dependency of the dimensions appears in the analysis (Fang et al., 2019; Jin et al., 2019).

1.6. Beyond first-order methods

Other researchers have contributed important work to approaches that do not rely solely on the gradient, but also use higher-order information. For instance, in (Agarwal et al., 2017), the *FastCubic* algorithm is based on Nesterov's cubic regularization and is capable of finding ϵ-second order stationary points in $\tilde{O}\left(\epsilon^{-7/4}\right)$. Although, these algorithms improve convergence analysis, they require Hessian information which may not be available in large scale machine learning problems.

Similar to the perturbed accelerated version of GD (Jin et al., 2017b), in (O'Neill & Wright, 2017; Sun et al., 2019), the authors analyze the behavior of accelerated methods such as the heavy-ball method when they are close to saddle points by using the stable manifold theorem proving that it is not very likely that accelerated methods get stuck in saddle points and that they can escape them faster than gradient-descent.

(Anandkumar & Ge, 2016) proposes the use of third order derivatives to escape degenerate saddle points. They also show that fourth order derivatives and higher is NP hard. Also, (Reddi et al., 2017) proposes a method that alternates between first-order and second-order subroutines to reduce the complexity of computing Hessians while allowing it to escape saddle points. The problem of escaping saddle points has also been explored in constraint optimization for quadratic objectives subject to convex sets (Mokhtari et al., 2018), as well as in the presence of Riemannian Manifolds (Criscitiello & Boumal, 2019; Sun et al., 2019).

Other important techniques are based on negative curvature information. For example, in (Xu et al., 2018; Allen-Zhu & Li, 2017) the authors propose NEON and NEON2, a methodology in which negative curvature information is extracted from the Hessian using only first order information attaining convergence rate of $\tilde{O}\left(\epsilon^{-4}\right)$ and $\tilde{O}\left(\epsilon^{-3.5}\right)$ respectively. Similarly, by adding regularization to the negative curvatures approach (Allen-Zhu, 2017) attains $\tilde{O}\left(\epsilon^{-3.25}\right)$ using Natasha2. Finally, using variance reduced gradient techniques SPIDER (Fang et al., 2018) is capable of achieving $\tilde{O}\left(\epsilon^{-3}\right)$ which is the current state of the art stochastic gradient computational cost.

1.7. Second order Stationarity in Machine Learning

In machine learning and signal processing non-convex problems it has been shown over the last years that all second-order stationary points are global minima, which means that if one can find second-order stationary points, this is equivalent to globally solving the problem. Areas where this has been shown include Tensor decomposition (Ge et al., 2015a), Dictionary Learning (Sun et al., 2016a), Phase Retrieval (Sun et al., 2016b), Synchronization and Max-Cut (Bandeira et al., 2016), Smooth Semidefinite Programs (Nicolas Boumal & Bandeira, 2016), Matrix sensing (Bhojanapalli et al., 2016), Matrix Completion (Ge et al., 2016) and Robust Principale Components (Rong Ge & Zheng, 2017). In most of these applications, the following is true:

1. All local minima are global minima
2. All saddle points have at least one direction with strictly negative curvature (are strict saddle points)

Note that for a function that meets 1 and 2, all second-order stationary points correspond to global minima. This observation further increases the importance of algorithms capable of finding second-order stationary points efficiently, as those reviewed here, since that guarantees global convergence in these non-convex problems.

One exciting area that has also received significant attention is that of Deep Learning. Several authors had been interested in the surface error of the training process of neural networks. (Dauphin et al., 2014) argues about the proliferation of saddle points when training a neural network and provides experimental evidence in this direction. Also, (Kawaguchi, 2016) provides a theoretical analysis showing that just like in other machine learning areas, in a Deep Network every local minimum is a global minimum and the remaining are saddle points. Furthermore, they argue that in deeper networks degenerate saddle points appear whereas strict saddle points are present in shallow networks (3 layers or less). This was an important result since it helped characterizing the optimization problem that needs to be solved in the training process of a neural network, showing that although it may be difficult it is easier than the general non-convex problem.

A more practical work by (Sankar & Balasubramanian, 2017) showed through an experimental setup that deep neural networks actually converge to saddle points and not to local minima and furthermore that these saddle points are degenerate. They claim that the theoretical work around convergence to saddle points only considers strict saddle points and therefore its motivation is questionable. We note that the definition of strict and degenerate saddle points between this work and the most notorious theoretical works (Ge et al., 2015b; Rong Ge & Zheng, 2017; Ge et al., 2016; Jin et al., 2017a; 2019) differs. For Sankar et al., (2017), degenerate saddle points can have positive, negative and zero eigenvalues of the Hessian. Furthermore, they call the number of zero eigenvalues the degree of degeneracy of the saddle point. For the theoretical works, a strict saddle point only requires the minimum eigenvalue of the Hessian to be negative, meaning that a saddle point with zero, positive and negative eigenvalues is still strict and can be analyzed under their framework.

In (Sankar & Balasubramanian, 2017), the authors characterize the critical point where the algorithms converge when training deep neural networks by explicitly computing the Hessian and its eigenvalue decomposition. Using this, they conclude that actually deep networks converge to saddle points and not to local minima. Although the conclusion somewhat agrees with other's authors observations, we point out that their experiments show one-layer neural networks that achieve comparable performance to deep networks due to the difficulty on computing the Hessian of a practical deep network. We believe that these conclusions can not be extracted from this approximation since the optimization problem may completely show different characteristics when using shallow and deep networks. It is widely known by the community that although shallow networks can match the performance of deep networks, this is not possible by using the gradient-based optimization algorithms that are usually used in this context.

In another work, (Daneshmand et al., 2018) show theoretically that when learning half-spaces using neural networks, the stochastic gradient has strong components in the direction of negative curvature. Using this observation, they propose a variation of PGD (Rong Ge & Zheng, 2017) where the noise is not added isotropic but the iteration is replaced by that of stochastic gradient descent. Their analysis show that this is enough to achieve convergence to second order stationary points.

Based on these theoretical and practical results and observations we propose to study the performance of perturbed gradient-based algorithms such as PGD and SPGD (Rong Ge & Zheng, 2017; Jin et al., 2019), in the training of deep neural networks.

2. Experimental Results

In this section, we will present preliminary results evaluating the perturbed version of stochastic gradient descent (PSGD), using the mini-batch method, versus vanilla stochastic gradient descent (SGD). The experimental setup is outlined at a high level in Fig. 2. This setup describes two main tests where PSGD and SGD convergence rate accuracy and loss are compared using relatively shallow and deep networks. The MLP architecture is considered a relatively shallow net-

work in the experiments due to it containing only 5 layers. Similarly, the VGG3 architecture is considered a relatively deep network because it contains 22 layers. The CIFAR-10 MLP has an input shape of (32,32,3) that is flattened. It then has a dense layer with 128 hidden neurons using sigmoid activation. This is followed by a 25% percent dropout, another dense 32 hidden neuron layer, and lastly 10 output neurons using softmax activation. The CIFAR-10 VGG3 has an input shape of (32,32,32) into a convolution layer followed by a max pooling and dropout layers. This layer sequence is repeated three times using Relu activation with the dropout increasing from 20% to 40%. The last four layers are flattening, dense 128 hidden neurons, 20% dropout, and 10 output neurons using softmax activation.

All networks were developed using Keras back-end onto TensorFlow. The networks were trained in Google Colab using Tesla K80 GPUs with 12 GB of RAM. Due to the execution time limitations set by Google Colab we were only able to run each experiment for 200 epochs. We used 50,000 images from the CIFAR-10 dataset for training and 10,000 for testing the network. In the following sections experimental results are presented for each algorithm based on the shallow and deep network tests depicted in the experimental setup.

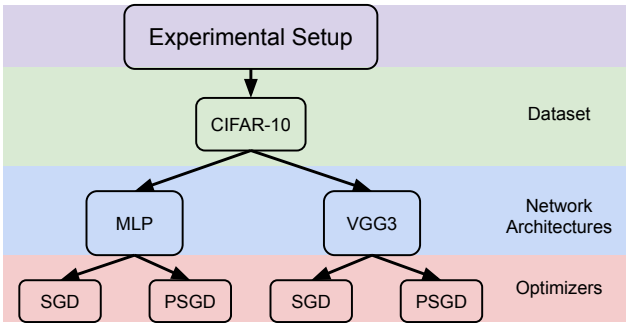


Figure 2. Experimental setup testing the performance of PSGD against SGD on relatively shallow (MLP) and deep (VGG3) networks.

2.1. Shallow Network Results

In this section both training and testing accuracy and loss results are presented for CIFAR10 using the MLP network architecture. The batch size was set to 128 and learning rate $\eta = 0.01$ for all experiments.

The training and test accuracy for SGD and PSGD, at various r values is shown in Fig.3-4 for 200 epochs. The corresponding training and testing loss for both algorithms is shown in Fig.5-6. Focusing on the test accuracy results in Fig.4, the maximum testing accuracy approaches 47% in 200 epochs when the PSGD algorithm is used with $r = 0.015$. When compared to the SGD algorithm the testing accuracy results show a significant increase in the convergence rate for the PSGD algorithm when $r = 0.015$. This increase starts at approximately 25 epochs and continues to converge exponentially faster than SGD with increased epochs. PSGD has a similar trend when $r = 0.1$ but after 150 epochs its convergence rate approximates that of SGD. The same results are present in the training data between PSGD and SGD as shown in Fig.3. However, in the training results the magnitude of convergence improvement in using PSGD with $r = 0.015$ is less pronounced when compared to the performance of SGD.

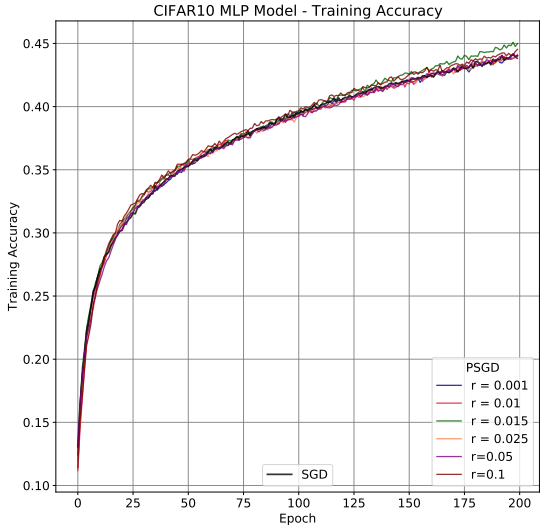


Figure 3. Training accuracy results for PSGD and SGD using a MLP network architecture. Implemented PSGD using mini-batch method described by Algorithm 5 at various r values. PSGD has moderate increase in convergence when $r = 0.1$ and significant increase when $r = 0.015$ compare to SGD. All other PSGD r values tested show negligible differences to SGD convergence rates.

In the testing loss results shown in Fig.6 there appears to be an inflection point around 25 epochs. The PSGD loss for all r values tested was greater than or equal to the SGD loss up to 25 epochs. After this point, up to 200 epochs, the PSGD algorithm with r values of 0.015 and 0.1 have significantly lower loss when compared to the SGD algorithm. When $r = 0.1$ the PSGD algorithm appears to have a fixed magnitude of lower loss compared to the SGD loss. However, when $r = 0.015$ the loss difference between PSGD and SGD increases in magnitude with increased epochs. The training loss shown in Fig.5 presents similar results to the testing

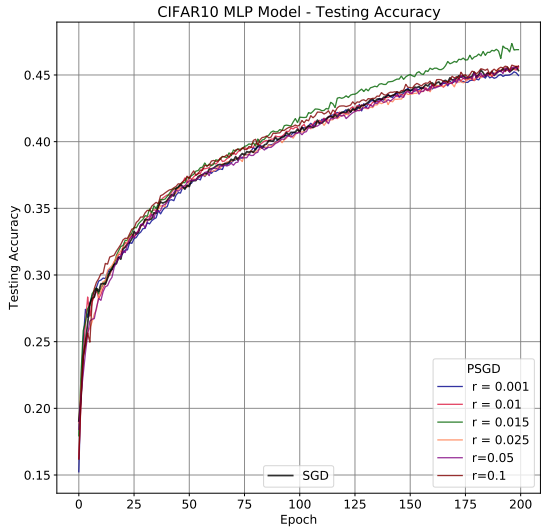


Figure 4. Testing accuracy results for PSGD and SGD using a MLP network architecture. Implemented PSGD using mini-batch method described by Algorithm 5 at various r values. PSGD has moderate increase in convergence when $r = 0.1$ between 10 -150 epochs and significant increase when $r = 0.015$ compare to SGD for all 200 epochs. All other PSGD r values tested show negligible differences to SGD convergence rates.

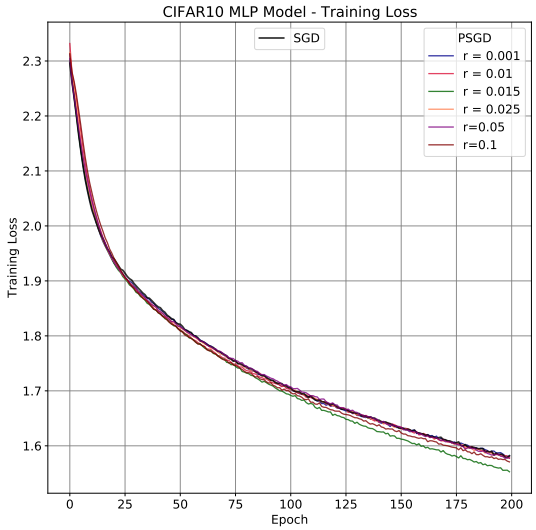


Figure 5. Training loss results for PSGD and SGD using a MLP network architecture. Implemented PSGD using mini-batch method described by Algorithm 5 at various r values. Comparing PSGD to SGD, it has a moderate decrease in L2 loss magnitude when $r = 0.1$ between 25 -150 epochs and significant decrease in L2 loss magnitude when $r = 0.015$ from 25 – 200 epochs. All other PSGD r values tested show negligible differences to SGD convergence rates.

loss except the PSGD algorithm with $r = 0.1$ starts to moderately converge to the SGD loss after 175 epochs.

Since the previous results shown in Figs.3-6 only represent data from one testing and training cycle, the improved PSGD performance could have been due to the random initialization. Thus, the training and testing process was repeated 10 times to validate that the improved convergence rate with $r = 0.015$ shown in Fig.4 was due to the PSGD algorithm. The average training and testing accuracy results for PSGD with $r = 0.015$ and SGD is shown in Figs.7-8. The corresponding average training and testing loss is shown in Figs.9-10.

The average testing accuracy in Fig.8 shows that the PSGD algorithm repeatably starts to converge faster than the SGD algorithm after approximately 75 epochs. After this point the standard deviation for both algorithms becomes very small supporting the notion that the PSGD algorithm with $r = 0.015$ is converging faster than the SGD algorithm in this experimental setup. The average training accuracy in Fig.7 shows a similar but not as distinguished trend as in the average testing accuracy results.

The average testing loss in Fig.9 supports the average accuracy results with the PSGD loss increasingly measuring a lower loss than SGD after 75 epochs. The significant difference between the PSGD and SGD average testing loss appears to be induced by algorithmic variations because the standard deviation in both results is negligible after 75 epochs. The average training loss in Fig.10 shows the same trend present in the average testing loss results.

Overall, the initial experimental results testing the shallow network in Figs.3-6 highlight the fact that the PSGD algorithm with $r = 0.015$ has significantly better convergence rate than the SGD algorithm for this experimental setup in 200 epochs. The average testing results for PSGD with $r = 0.015$ and SGD shown in Figs.7-10 suggest this convergence increase is partially algorithmically induced. Previous work by Sankar and Balasubramanian (2017) studied the relationship between saddle point degeneracy and neural networks and proposed that convergence points of deep neural networks tend to be saddle points that increase in degeneracy with increased depth (Sankar & Balasubramanian, 2017). Recall that the MLP architecture is 5 layers deep which means it could contain many low degenerate saddle points. However, during training if the gradient descent method converged to a saddle point we should see it plateau

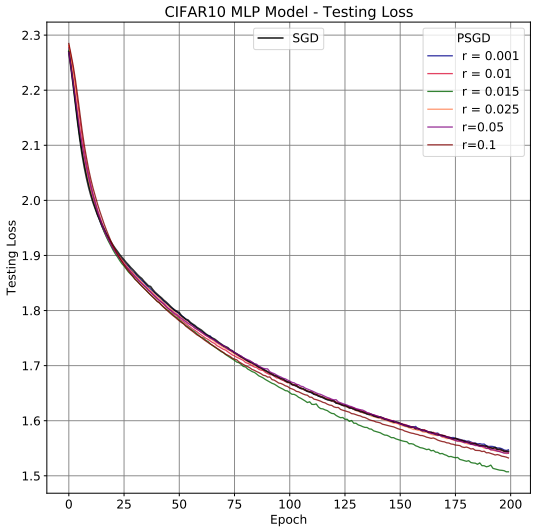


Figure 6. Testing loss results for PSGD and SGD using a MLP network architecture. Implemented PSGD using mini-batch method described by Algorithm 5 at various r values. PSGD shows the same trends, just more pronounced as presented in the training loss results shown in Fig.5.

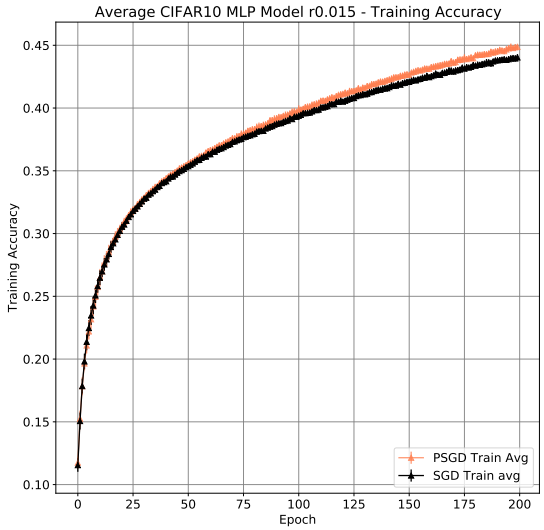


Figure 7. Training accuracy for PSGD using mini-batch method described in Algorithm 5 at r = 0.015 for 10 trials and compared to a single SGD trial. PSGD has an average convergence rate that is moderately improved compared to the SGD average convergence rate.

in accuracy or loss per iteration. It is unclear in the 200 epoch presented results if the SGD algorithm and all other PSGD algorithms with r values other than 0.015 are converging to a degenerate saddle point. If that is the case then based on this experimental setup setting the r magnitude to 0.015 seems to optimally tune the high dimensional zero-mean Gaussian distribution that the gradient perturbation is sampled from in the PSGD algorithm. However, these results are preliminary and require further testing before a conclusion about the practicality of the analyzed algorithms can be made.

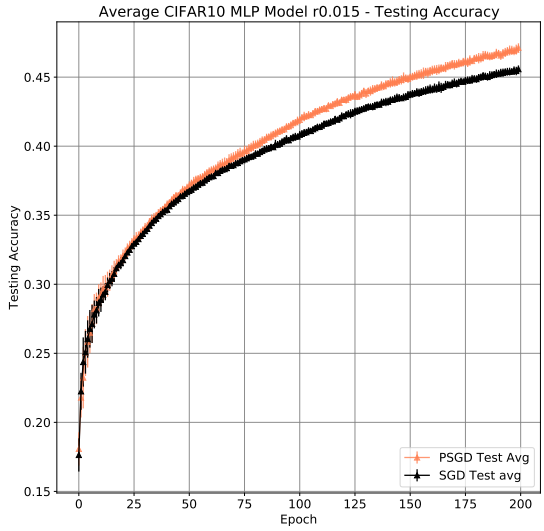


Figure 8. Testing accuracy for PSGD using mini-batch method described in Algorithm 5 at r = 0.015 for 10 trials and compared to a single SGD trial. PSGD has an average convergence rate that is significantly improved compared to the SGD average convergence rate between 75-200 epochs. The PSGD convergence rate is increasingly improving with each epoch compared to SGD.

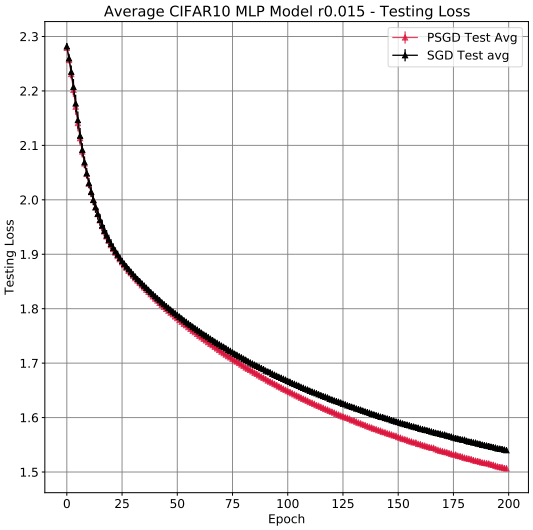


Figure 10. Testing loss for PSGD using mini-batch method described in Algorithm 5 at r = 0.015 for 10 trials and compared to a single SGD trial. PSGD has an average ℓ_2 loss that is significantly improved compared to the SGD average ℓ_2 loss between 75-200 epochs. The PSGD ℓ_2 loss increasingly lowers per epoch compared to the SGD ℓ_2 loss.

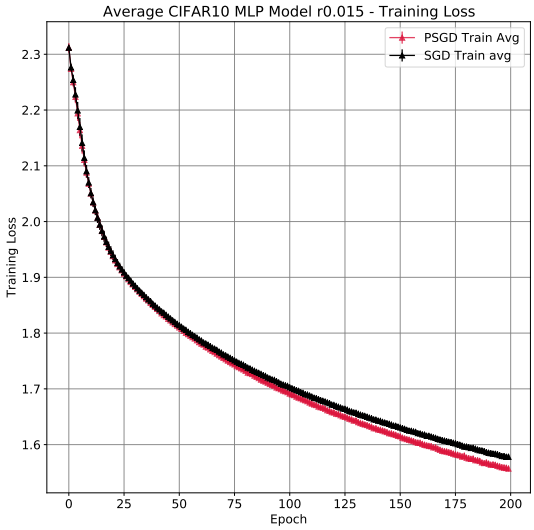


Figure 9. Training loss for PSGD using mini-batch method described in Algorithm 5 at r = 0.015 for 10 trials and compared to a single SGD trial. PSGD has an average ℓ_2 loss is moderately improved compared to the SGD average ℓ_2 loss.

2.2. Deep Network Results

In this section both training and testing accuracy and loss results are presented for CIFAR10 using the VGG3 network architecture. The batch size was set to 128 and learning rate $\eta = 0.001$.

The training accuracy over 200 epoch for SGD and PSGD at various r values is shown in Fig.11. The corresponding testing accuracy is shown for the same case in Fig.12. The training and testing loss are presented in Figs.13-14 respectively. The general trend throughout the accuracy plots in Figs.11-12 is increasing the parameter r decreases the convergence rate and overall performance of the network when trained using the PSGD optimizer. In this setup the maximum testing accuracy in 200 epochs approaches 70% when using the SGD optimizer. The SGD optimizer appears to be an upper bound on the performance of the PSGD optimizer during this 200 epoch range with this specific experimental setup. However, the only immediate exception to this claim is when PSGD has the r parameter set to 0.025. At this setting PSGD converges faster then SGD between 20-100 epochs and then converges slower than SGD after 130 epochs.

Further tests were run using PSGD with $r = 0.025$ to verify that the slightly faster convergence compared to SGD, shown in Figs.11-12, was due to the PSGD algorithm and not the random initialization point. The results in Figs.15-16 show the training and testing accuracy for 10 PSGD optimized trials averaged and compared to SGD on the CIFAR-10 dataset using the VGG3 architecture. In Fig.16 the testing accuracy results show that the PSGD algorithm convergence rates have significant varaince between 20-100 epochs and the average approximates the convergence performance of the SGD algorithm. This means that in this specific experimental setup the PSGD algorithm has at a maximum the same performance as SGD both in convergence rate and accuracy.

The training and testing loss plots in Figs.13-14 where PSGD for various r values is compared to SGD, depicts a similar trend as seen in the accuracy plots for this test. The loss magnitude for both training and testing generally increases with and increased r value. Analogous to the accuracy results, the average loss of PSGD with $r = 0.025$ for 10 trials shown in Figs.17-18 support the findings that the PSGD algorithm does not outperform the traditional SGD algorithm for this experimental setup.

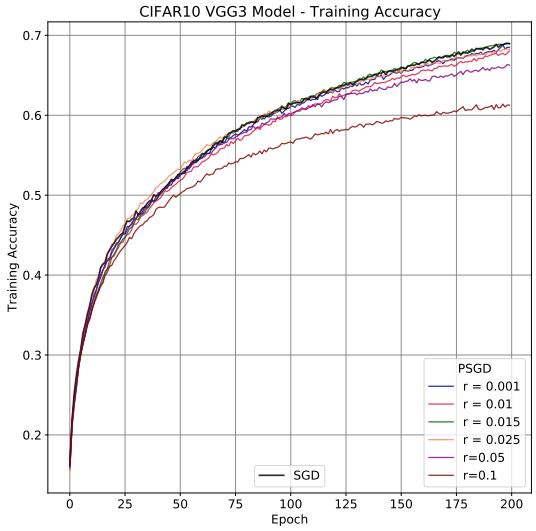


Figure 11. Implemented PSGD using mini-batch method described in Algorithm 5 at various r values. Training accuracy convergence rate is similar to SGD with increasing r decreasing PSGD convergence rate.

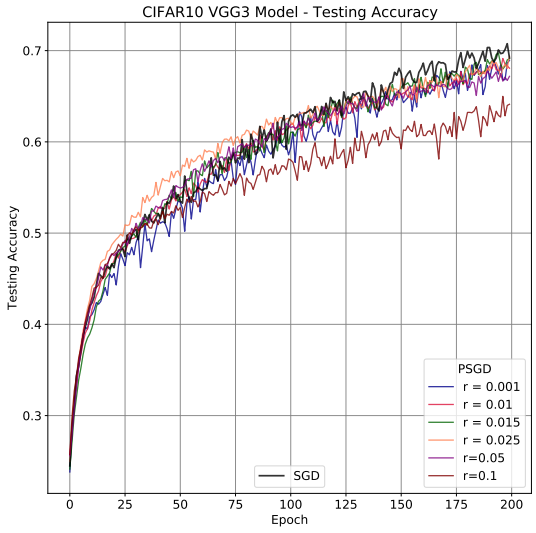


Figure 12. Implemented PSGD using mini-batch method described in Algorithm 5 at various r values. Testing accuracy convergence rate is similar to SGD with increasing r decreasing PSGD convergence rate. Case of PSGD with $r = 0.025$ appears to have improved convergence between 20-100 epochs but this is due to initialization point.

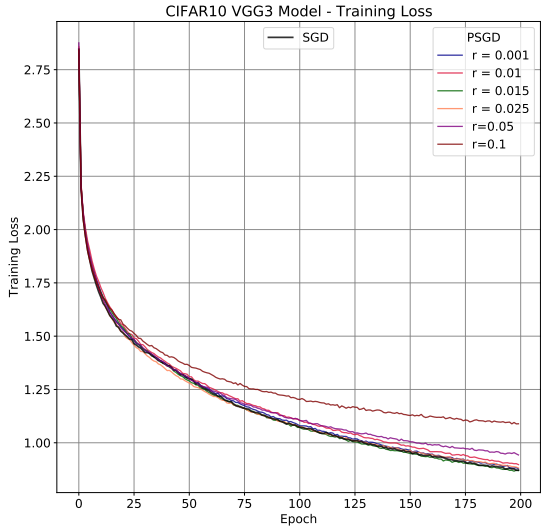


Figure 13. Implemented PSGD using mini-batch method described in Algorithm 5 at various r values. Training loss is similar to SGD with increasing r increasing PSGD loss magnitude.

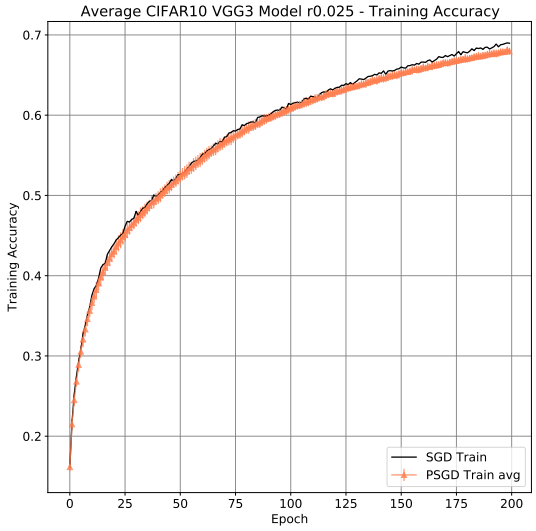


Figure 15. Training accuracy for PSGD using mini-batch method described in Algorithm 5 at $r = 0.025$ for 10 trials and compared to a single SGD trial.

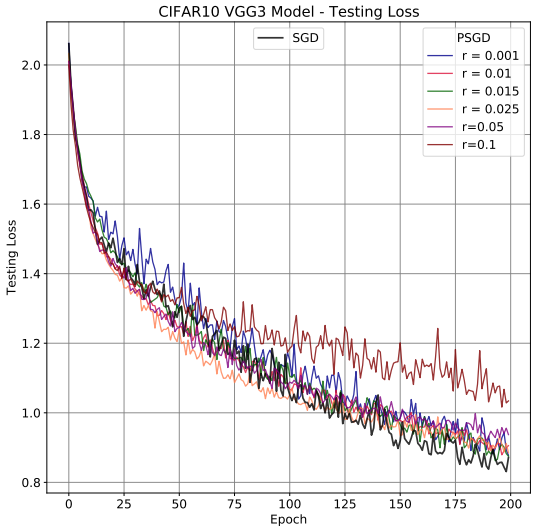


Figure 14. Implemented PSGD using mini-batch method described in Algorithm 5 at various r values. Testing loss is similar to SGD with increasing r increasing PSGD loss magnitude. Case of PSGD with $r = 0.025$ appears to have improved loss between 20-100 epochs but this is due to initialization point.

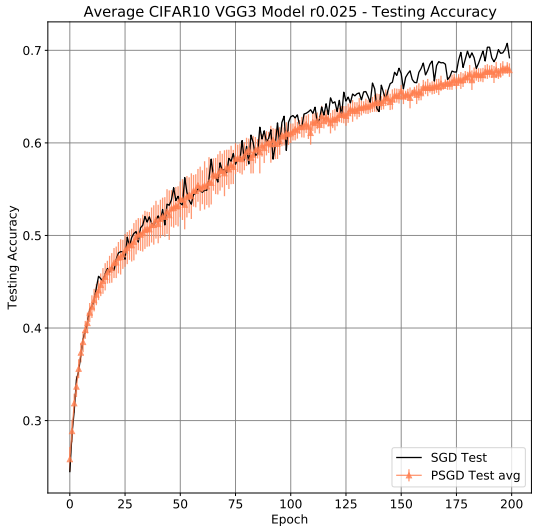


Figure 16. Testing accuracy for PSGD using mini-batch method described in Algorithm 5 at $r = 0.025$ for 10 trials and compared to a single SGD trial. Testing accuracy shows high variance between 20-100 epochs with the average approximating the SGD convergence result.

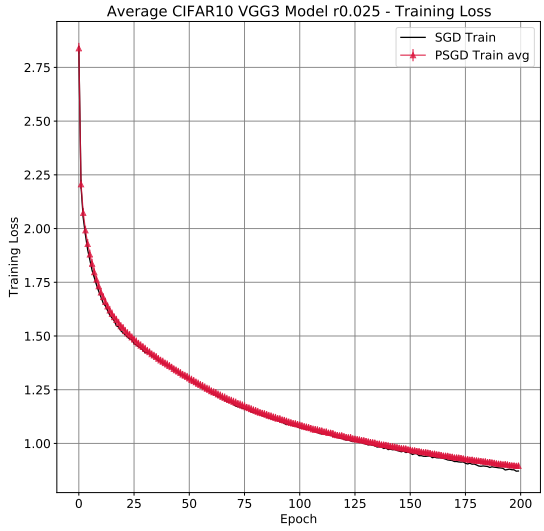


Figure 17. Training loss for PSGD using mini-batch method described in Algorithm 5 at $r = 0.025$ for 10 trials and compared to a single SGD trial.

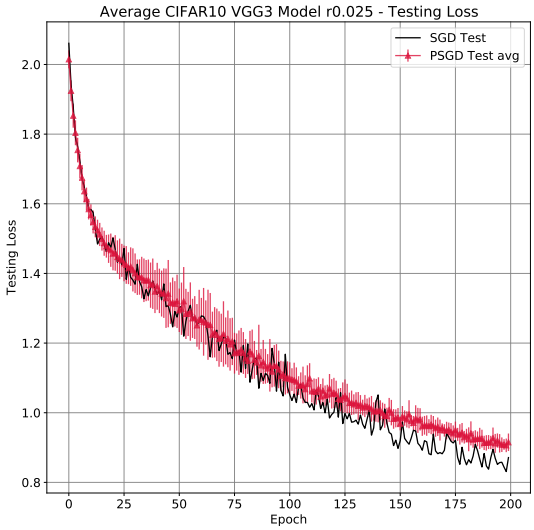


Figure 18. Testing loss for PSGD using mini-batch method described in Algorithm 5 at $r = 0.025$ for 10 trials and compared to a single SGD trial. Testing accuracy shows high variance between 20-100 epochs with the average approximating the SGD convergence result.

3. Conclusion

This work provides a comprehensive literature review of gradient based algorithms designed to efficiently escape saddle points in non-convex settings. The literature review originated from (Jin et al., 2017a) work "How to Escape Saddle Points Efficiently?". The authors derived a gradient descent variant algorithm called perturbed gradient descent (PGD) that theoretically will converge to a ϵ-second order stationary point in a number of iterations that is almost "dimension-free". A majority of recent work is focused on improving the convergence rate of gradient based algorithms that can still escape strict saddle points efficiently. Future work could be focused on achieving third-order stationary points in the presence of saddle points and determining the GD convergence rates for this condition (Jin et al., 2019). However, significant applications of when third-order stationary points are beneficial will also need to be established.

This work also provides experimental convergence results for perturbed versions of gradient descent on different MLP and VGG3 network architectures using the CIFAR-10 dataset. This is some of the first experimental results testing some of the current proposed theoretical perturbed gradient descent techniques on neural networks. Initial results show an improved experimental convergence rate for perturbed stochastic gradient descent with $r = 0.015$ between 10-150 epochs when compared to general SGD using a MLP architecture. There was no significant improvement to using a perturbed version of SGD with the VGG3 architecture. While results are promising they are preliminary and further experimentation need to be conducted before a conclusion can be made on the practicality of the perturbed version of SGD.

The next stage for these experiments is to run the tests for an increased number of epochs with access to GPUs. These experiments could be further extended to investigate the following open ended questions: possibility of adding scheduled perturbations, auto-detection for adding perturbation when necessary, determining what type of perturbation (uniform dimensional ball or normal distribution) is the best to add. Also investigating ways to use randomized algorithms to approximate the minimized eigenvalue of the Hessian; to help characterize the type of convergence point the algorithms are converging to during each experiment.

References

Agarwal, N., Allen-Zhu, Z., Bullins, B., Hazan, E., and Ma, T. Finding approximate local minima faster than gradient descent, 2017.

Allen-Zhu, Z. Natasha 2: Faster non-convex optimization than sgd, 2017.

Allen-Zhu, Z. and Li, Y. Neon2: Finding local minima via first-order oracles. *CoRR*, abs/1711.06673, 2017. URL http://arxiv.org/abs/1711.06673.

Anandkumar, A. and Ge, R. Efficient approaches for escaping higher order saddle points in non-convex optimization. *CoRR*, abs/1602.05908, 2016. URL http://arxiv.org/abs/1602.05908.

Bandeira, A. S., Boumal, N., and Voroninski, V. On the low-rank approach for semidefinite programs arising in synchronization and community detection. *Conference on Learning Theory*, pp. 361–382, 2016.

Bhojanapalli, S., Neyshabur, B., , and Srebro, N. Global optimality of local search for low rank matrix recovery. *Advances in Neural Information Processing Systems*, pp. 3873–3881, 2016.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. The loss surface of multilayer networks, 2014. URL arXiv:1412.0233.

Criscitiello, C. and Boumal, N. Efficiently escaping saddle points on manifolds, 2019.

Daneshmand, H., Kohler, J. M., Lucchi, A., and Hofmann, T. Escaping saddles with stochastic gradients. *CoRR*, abs/1803.05999, 2018. URL http://arxiv.org/abs/1803.05999.

Dauphin, Y. N., Pascanu, R., Gülçehre, Ç., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2572, 2014. URL http://arxiv.org/abs/1406.2572.

Du, S. S., Jin, C., Lee, J. D., Jordan, M. I., Poczos, B., and Singh, A. Gradient descent can take exponential time to escape saddle points, 2017.

Fang, C., Li, C. J., Lin, Z., and Zhang, T. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 689–699. Curran Associates, Inc., 2018.

Fang, C., Lin, Z., and Zhang, T. Sharp analysis for nonconvex sgd escaping from saddle points, 2019.

Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points - online stochastic gradient for tensor decomposition. *arXiv: 1503.02101v1 [cs.LG]*, pp. 1–46, March 2015a.

Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points — online stochastic gradient for tensor decomposition. In Grünwald, P., Hazan, E., and Kale, S. (eds.), *Proceedings of The 28th Conference on Learning Theory*, volume 40 of *Proceedings of Machine Learning Research*, pp. 797–842, Paris, France, 03–06 Jul 2015b. PMLR. URL http://proceedings.mlr.press/v40/Ge15.html.

Ge, R., Lee, J. D., and Ma, T. Matrix completion has no spurious local minimum. *Advances in Neural Information Processing Systems*, pp. 2973–2981, 2016.

Ghadimi, S. and Lan, G. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013. doi: 10.1137/120880811. URL https://doi.org/10.1137/120880811.

Jin, C., Ge, R., Netrapalli, P., Kakade, S. M., and Jordan, M. I. How to escape saddle points efficiently. *arXiv: 1703.00887v1 [cs.LG]*, pp. 1–35, March 2017a.

Jin, C., Netrapalli, P., and Jordan, M. I. Accelerated gradient descent escapes saddle points faster than gradient descent. *CoRR*, abs/1711.10456, 2017b. URL http://arxiv.org/abs/1711.10456.

Jin, C., Ge, R., Netrapalli, P., Ge, R., Kakade, S. M., and Jordan, M. I. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points. *arXiv: 1902.04811v2 [cs.LG]*, pp. 1–31, September 2019.

Kawaguchi, K. Deep learning without poor local minima, 2016.

Lee, J. D., Simchowitz, M., Jordan, M. I., and Recht, B. Gradient descent converges to minimizers, 2016.

Levy, K. Y. The power of normalization: Faster evasion of saddle points. *CoRR*, abs/1611.04831, 2016. URL http://arxiv.org/abs/1611.04831.

Mokhtari, A., Ozdaglar, A. E., and Jadbabaie, A. Escaping saddle points in constrained optimization. *CoRR*, abs/1809.02162, 2018. URL http://arxiv.org/abs/1809.02162.

Nesterov, Y. *Introductory lectures on convex programming volume i: Basic course*.

Nicolas Boumal, V. V. and Bandeira, A. The non-convex burer-monteiro approach works on smooth semidefinite programs. *Advances in Neural Information Processing Systems*, pp. 2757–2765, 2016.

O'Neill, M. and Wright, S. J. Behavior of accelerated gradient methods near critical points of nonconvex functions, 2017.

Reddi, S. J., Zaheer, M., Sra, S., Póczos, B., Bach, F. R., Salakhutdinov, R., and Smola, A. J. A generic approach for escaping saddle points. *CoRR*, abs/1709.01434, 2017. URL http://arxiv.org/abs/1709.01434.

Rong Ge, C. J. and Zheng, Y. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. *arXiv preprint arXiv:1704.00708*, 2017.

Sankar, A. R. and Balasubramanian, V. N. Are saddles good enough for deep learning?, 2017.

Sun, J., Qu, Q., and Wright., J. Complete dictionary recovery over the sphere i: Overview and the geometric picture. *IEEE Transactions on Information Theory*, 2016a.

Sun, J., Qu, Q., and Wright., J. A geometric analysis of phase retrieval. *2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 2379–2383, 2016b.

Sun, T., Li, D., Quan, Z., Jiang, H., Li, S., and Dou, Y. Heavy-ball algorithms always escape saddle points, 2019.

Xu, Y., Jin, R., and Yang, T. First-order stochastic algorithms for escaping from saddle points in almost linear time, 2018.